C. ABDUL HAKEEM COLLEGE OF ENGINEERING AND TECHNOLOGY,
Hakeem Nagar, Melvisharam - 632 509, Ranipet District, Tamil Nadu, India.
(Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai)
(Regd. Under Sec 2(F) & 12(B) of the UGC Act 1956)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

AD3461 – MACHINE LEARNING LABORATORY RECORD

(REGULATION - 2021)

Name of the Student:

Register Number:

Degree / Branch:

Year / Semester:

Academic Year:

C. ABDUL HAKEEM COLLEGE OF ENGINEERING AND TECHNOLOGY

Hakeem Nagar, Melvisharam - 632 509, Ranipet District, Tamil Nadu,India.

(Approved by AICTE, New Delhi and Affiliated to Anna University,Chennai)

Regd. Under Sec 2(F) & 12(B) of the UGC Act 1956)

Name of the Candidate:

Year: II **Semester:** IV **Degree/Branch:** B. TECH/AI&DS

Subject Code: AD3461
Subject Name: MACHINE LEARNING LABORATORY

University Register Number:

CERTIFICATE

Certified that this is the bonafide record of work done by the above student in

AD3461 – MACHINE LEARNING LABORATORY

during 2023 - 2024.

Signature of Head of the Department **Signature of Lab In-charge**

Submitted for the University Practical Examination held on ______________

EXAMINERS

Date:__________________ **Centre code:** 5106

Internal:__________________________ **External:**________________________

INDEX

Ex.No:1
Date:

FIND – S Algorithm

Aim:

To implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on training data samples. Read the training data from a .CSV file.

Algorithm:

```
Step 1: Initialize h to the most specific hypothesis in H
Step 2: For each positive training instance x
    Step 2.1: For each attribute constraint ai in h
        Step 2.1.1: If the constraint $a_i$ is satisfied by x
            Step 2.1.1.1: Then do nothing
        Step 2.1.2: Else replace $a_i$ in h with the next more general constraint
                satisfied by x.
Step 3: Output hypothesis h
```

Training Examples:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import csv
a = []

with open('C:/Users/Desktop/ML/lab/enjoysport.csv', 'r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)
    print(a)

print("\n The total number of training instances are : ",len(a))

num_attribute = len(a[0])-1

print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)

for i in range(0, len(a)):
    if a[i][num_attribute] == 'yes':
        for j in range(0, num_attribute):
            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
            else:
                hypothesis[j] = '?'
    print("\n The hypothesis for the training instance {} is : \n"
.format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instance
is ")
print(hypothesis)
```

Output:

```
[['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast',
'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']]

 The total number of training instances are :  5

 The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 1 is :
 ['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 2 is :
 ['Sunny', 'Warm', 'Normal', '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', 'Warm', 'Same']

 The hypothesis for the training instance 5 is :
 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 The Maximally specific hypothesis for the training instance is
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Result:

Thus, the Python code to implement the FIND -S Algorithm was executed and trained using the dataset, and the output was verified successfully.

Ex.No:2 **CANDIDATE ELIMINATION Algorithm**
Date:

Aim:

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.

Algorithm:

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

```
Step 1: Initialize G to the set of maximally general hypotheses in H.
Step 2: Initialize S to the set of maximally specific hypotheses in H.
Step 3: For each training example d, do
    Step 3.1: If d is a positive example.
        Step 3.1.1: Remove from G any hypothesis inconsistent with d.
        Step 3.1.2: For each hypothesis s in S that is not consistent with d
            Step 3.1.2.1: Remove s from S.
            Step 3.1.2.2: 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.
            Step 3.1.2.3: Remove from S any hypothesis that is more general
                          than another hypothesis in S.
    Step 3.2: If d is a negative example.
        Step 3.2.1: Remove from S any hypothesis inconsistent with d.
        Step 3.2.2: For each hypothesis g in G that is not consistent with d.
            Step 3.2.2.1: Remove g from G.
            Step 3.2.2.2: 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.
            Step 3.2.2.3: Remove from G any hypothesis that is less general than
                          another hypothesis in G.
```

Training Examples:

Example	**Sky**	**AirTemp**	**Humidity**	**Wind**	**Water**	**Forecast**	**EnjoySport**
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import numpy as np
import pandas as pd

data = pd.read_csv(r"C:\Users\Desktop\ML\lab\enjoysport.csv")
concepts = np.array(data.iloc[:, 0:-1])
print(concepts)
target = np.array(data.iloc[:, -1])
print(target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print(general_h)

    for i, h in enumerate(concepts):
        print("For Loop Starts")
        if target[i] == "yes":
            print("If 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("If 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(" steps of Candidate Elimination Algorithm", i + 1)
        print(specific_h)
        print(general_h)
        print("\n")
        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")
```

Output:

```
[[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 [2 'Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 [3 'Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 [4 'Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
['Yes' 'Yes' 'No' 'Yes']
initialization of specific_h and general_h
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]
For Loop Starts
 steps of Candidate Elimination Algorithm 1
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]

For Loop Starts
 steps of Candidate Elimination Algorithm 2
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]

For Loop Starts
 steps of Candidate Elimination Algorithm 3
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]

For Loop Starts
 steps of Candidate Elimination Algorithm 4
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]

Final Specific_h:
[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Final General_h:
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]
```

Result:

Thus, the Python code to implement the Candidate Elimination Algorithm was executed and trained using the dataset and the output was verified successfully.

Ex.No:3 **ID3 Algorithm**
Date:

Aim:

To write a program to demonstrate the working of the decision tree based ID3 algorithm. Using an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Algorithm ID3(Examples, Target_attribute, Attributes)

```
Step 1: Create a Root node for the tree
Step 2: If all Examples are positive, Return the single-node tree Root, with label = +
Step 3: If all Examples are negative, Return the single-node tree Root, with label = -
Step 4: If Attributes is empty, Return the single-node tree Root, with label = most
        common valueof Target_attribute in Examples
Step 5: Otherwise Begin
    Step 5.1: A ← the attribute from Attributes that best* classifies Examples
    Step 5.2: The decision attribute for Root ← A
    Step 5.3: For each possible value, vi, of A,
     Step 5.3.1: Add a new tree branch below Root, corresponding to the test A = vi
     Step 5.3.2: Let Examples vi, be the subset of Examples that have value vi for A
     Step 5.3.3: If Examples vi , is empty
          Step 5.3.3.1: Then below this new branch add a leaf node with
                  label = most commonvalue of Target_attribute in Examples
          Step 5.3.3.2: Else below this new branch add the subtree
                  ID3(Examples vi, Target_attribute, Attributes – {A}))
Step 6: End.
Step 7: Return Root.
```

ENTROPY:

Entropy measures the impurity of a collection of

$$Entropy\,(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

examples.

Where, *p+* is the proportion of positive examples in S

p- is the proportion of negative examples in S.

INFORMATION GAIN:

- ***Information gain,*** is the expected reduction in entropy caused by partitioning theexamples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examplesS, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Training Dataset:

Day	***Outlook***	***Temperature***	***Humidity***	***Wind***	***PlayTennis***
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 Dataset:

Day	***Outlook***	***Temperature***	***Humidity***	***Wind***
T1	Rain	Cool	Normal	Strong
T2	Sunny	Mild	Normal	Strong

Program:

```
import math
import csv

def load_csv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset, headers

class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

def subtables(data, col, delete):
    dic = {}
    coldata = [row[col] for row in data]
    attr = list(set(coldata))

    counts = [0] * len(attr)
    r = len(data)
    c = len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col] == attr[x]:
                counts[x] += 1

    for x in range(len(attr)):
        dic[attr[x]] = [[0 for i in range(c)] 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]
                dic[attr[x]][pos] = data[y]
                pos += 1
    return attr, dic

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
```

```python
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

def build_tree(data, features):
    lastcol = [row[-1] for row in data]
    if (len(set(lastcol))) == 1:
        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

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)

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(r"C:\Users\Desktop\ML\labid3.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(r"C:\Users\Desktop\ML\lab\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
     Overcast
         Yes
     Sunny
         Humidity
            Normal
               Yes
            High
               No
     Rainy
         Wind
            Weak
               Yes
            Strong
               No
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
```

Result:

Thus, the Python code to implement Decision Tree – based ID3 Algorithm was executed and trained using dataset and the output was verified successfully.

Ex.No:4 **ANN using Backpropagation**
Date:

Aim:

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

Algorithm:

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

Each training example is a pair of the form $(\vec{x}, \vec{t})$, where $(\vec{x})$ is the vector of network input values, $(\vec{t})$ and is the vector of target network output values.
η is the learning rate (e.g., .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}

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

    - For each $(\vec{x}, \vec{t})$, in training examples, Do

      Propagate the input forward through the network:
      1. Input the instance $\vec{x}$ 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 output unit k, calculate its error term δ_k

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

      3. For each hidden 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

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

Training Examples:

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

<u>Normalize the input</u>

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

<u>Program:</u>

```
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:
 [[0.8958323 ]
 [0.87642037]
 [0.89721966]]
```

Result:

Thus, the Python code to build an Artificial Neural Network by implementing the Backpropagation algorithm was trained and executed and the output was verified successfully.

Ex.No: 5 **Naïve Bayesian Classifier**
Date:

Aim:
To implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering a few test data sets.

Algorithm:
```
Step 1: Data Handling.
     Step 1.1: Loading the Data from csv file of the Pima indians diabetes
dataset.
     Step 1.2: Splitting the Data set into Training Set.
Step 2.Summarize Data.
       Step 2.1: Separate Data By Class.
       Step 2.2: Calculate Mean.
       Step 2.3: Calculate Standard Deviation.
       Step 2.4: Summarize Dataset.
       Step 2.5: Summarize Attributes By Class.
Step 3: Make Prediction.
       Step 3.1: Calculate Probability Density Function.
       Step 3.2: Calculate Class Probabilities.
       Step 3.3: Find out the largest probability and return the associated class.
Step 4: Make Predictions.
      Step 4.1: Prepare Model.
      Step 4.2: Test Model.
      Step 4.3: Print the Result and Accuracy.
```
Procedure:
- Compute the prior probability for the target class.
- Compute the frequency matrix and likelihood probability for each feature.
- Use Bayes theorem to calculate the probability of all hypotheses.

Bayesian Theorem:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- $P(h)$ = prior probability of hypothesis h
- $P(D)$ = prior probability of training data D
- $P(h|D)$ = probability of h given D
- $P(D|h)$ = probability of D given h

- Classify the test object using the Maximum A Posteriori (MAP) Hypothesis.

$$\text{Posterior} = \frac{\text{Likelihood x Prior}}{\text{Evidence}}$$

Naive Bayes: For the Bayesian Rule above, we have to extend it so that we have

$$P(C|X_1, X_2, \ldots, X_n) = \frac{P(X_1, X_2, \ldots, X_n|C)\, P(C)}{P(X_1, X_2, \ldots, X_n)}$$

Bayes' rule:

Given a set of variables, X = {x1,x2,x...,xd}, we want to construct the posterior probability for the event Cj among a set of possible outcomes C = {c1,c2,c...,cd} , the Bayes Rule is

$$p(C_j \mid x_1, x_2, \ldots, x_d) \propto p(x_1, x_2, \ldots, x_d \mid C_j) p(C_j)$$

Since Naive Bayes assumes that the conditional probabilities of the independent variables are statistically independent, we can decompose the likelihood to a product of terms:

$$p(X \mid C_j) \propto \prod_{k=1}^{d} p(x_k \mid C_j)$$

and rewrite the posterior as:

$$p(C_j \mid X) \propto p(C_j) \prod_{k=1}^{d} p(x_k \mid C_j)$$

Using Bayes' rule above, we label a new case X with a class level Cj that achieves the highest posterior

probability.

Naive Bayes can be modeled in several different ways including normal, lognormal, gamma and Poisson density functions:

$$p(x_k \mid C_j) = \begin{cases} \dfrac{1}{\sigma_{kj}\sqrt{2\pi}} \exp\left(\dfrac{-(x-\mu_{kj})^2}{2\sigma_{kj}}\right), \quad -\infty < x < \infty, -\infty < \mu_{kj} <, \sigma_{kj} > 0 & \text{Normal} \\ \mu_{kj}: \text{mean}, \ \sigma_{kj}: \text{standard deviation} & \\ \dfrac{1}{x\sigma_{kj}(2\pi)^{1/2}} \exp\left\{\dfrac{-[\log(x/m_{kj})]^2}{2\sigma_{kj}^2}\right\}, \quad 0 < x < \infty, m_{kj} > 0, \sigma_{kj} > 0 & \text{Lognormal} \\ m_{kj}: \text{scale parameter}, \ \sigma_{kj}: \text{shape parameter} & \\ \dfrac{\left(\dfrac{x}{b_{kj}}\right)^{c_{kj}-1}}{b_{kj}\Gamma(c_{kj})} \exp\left(\dfrac{-x}{b_{kj}}\right), \quad 0 \le x < \infty, b_{kj} > 0, c_{kj} > 0 & \text{Gamma} \\ b_{kj}: \text{scale parameter}, \ c_{kj}: \text{shape parameter} & \\ \dfrac{\lambda_{kj} \exp(-\lambda_{kj})}{x!}, \quad 0 \le x < \infty, \lambda_{kj} > 0, x = 0, 1, 2, \ldots & \text{Poisson} \\ \lambda_{kj}: \text{mean} & \end{cases}$$

Types

• Gaussian: It is used in classification and it assumes that features follow a normal distribution.

Gaussian Naive Bayes is used in cases when all our features are continuous. For example in Iris dataset features are sepal width, petal width, sepal length, petal length

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

- Multinomial Naive Bayes : Its is used when we have discrete data (e.g. movie ratings ranging 1 and 5 as each rating will have certain frequency to represent). In text learning we have the count of each word to predict the class or label

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}{}^{x_i}$$

$$\hat{P}(x_i \mid \omega_j) = \frac{\sum tf(x_i, d \in \omega_j) + \alpha}{\sum N_{d \in \omega j} + \alpha \cdot V}$$

- Bernoulli Naive Bayes: It assumes that all our features are binary such that they take only two values. This means 0s can represent “word does not occur in the document” and 1s as "word occurs in the document"

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

Examples:

- The data set used in this program is the ***Pima Indians Diabetes problem***.
- This data set is comprised of 768 observations of medical details for Pima Indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.
- The attributes are Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome.
- Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)

Sample Examples:

Examples	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1

Program:

```
import csv
import random
import math

def loadcsv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    for i in range(len(dataset)):
        # converting strings into numbers for processing
        dataset[i] = [float(x) for x in dataset[i]]

    return dataset

def splitdataset(dataset, splitratio):
    # 67% training size
    trainsize = int(len(dataset) * splitratio);
    trainset = []
    copy = list(dataset);
    while len(trainset) < trainsize:
        # generate indices for the dataset list randomly to pick ele
for training data
        index = random.randrange(len(copy));
        trainset.append(copy.pop(index))
    return [trainset, copy]

def separatebyclass(dataset):
    separated = {}  # dictionary of classes 1 and 0
    # creates a dictionary of classes 1 and 0 where the values are
    # the instances belonging to each class
    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):
    return sum(numbers) / float(len(numbers))

def stdev(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):  # creates a dictionary of classes
    summaries = [(mean(attribute), stdev(attribute)) for attribute in
zip(*dataset)];
    del summaries[-1]  # excluding labels +ve or -ve
    return summaries

def summarizebyclass(dataset):
    separated = separatebyclass(dataset);
    # print(separated)
    summaries = {}
    for classvalue, instances in separated.items():
        # for key,value in dic.items()
        # summaries is a dic of tuples(mean,std) for each class value
        summaries[classvalue] = summarize(instances)   # summarize is
used to cal to mean and std
    return summaries

def calculateprobability(x, mean, stdev):
    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):
    probabilities = {}  # probabilities contains the all prob of all
class of test data
    for classvalue, classsummaries in summaries.items():  # class and
attribute information as mean and sd
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i]  # take mean and sd of
every attribute for class 0 and 1 seperaely
            x = inputvector[i]  # testvector's first attribute
            probabilities[classvalue] *= calculateprobability(x, mean,
stdev);  # use normal dist
    return probabilities

def predict(summaries, inputvector):  # training and test data is
passed
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():  # assigns
that class which has he highest prob
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel

def getpredictions(summaries, testset):
    predictions = []
    for i in range(len(testset)):
        result = predict(summaries, testset[i])
        predictions.append(result)
    return predictions

def getaccuracy(testset, predictions):
    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 = 'C:/Users/olgar/OneDrive/Desktop/ML/lab/my
manual/Exp5/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);
    # print(summaries)
    # test model
    predictions = getpredictions(summaries, testset)  # find the
predictions of test data with the training data
    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 : 75.59055118110236%
```

Result:

Thus, the Python code to implement the naïve Bayesian classifier for a sample training data set was executed and the output was verified successfully.

Ex.No: 6 **Naïve Bayesian Classifier (documents)**
Date:

Aim:
To implement the naïve Bayesian classifier to classify a set of documents using Python code and also to calculate the accuracy, precision and recall for the chosen dataset.

Algorithm:

STEP 1: Convert the training dataset into a frequency table where each row represents a document, and each column represents a word in the vocabulary. The values in the table represent the frequency of each word in each document.

STEP 2: Calculate the prior probabilities of each class label by diving the number of documents in each class by the total number of documents.

STEP 3: Calculate the conditional probabilities of each word given each class label. This involves calculating the frequency of each word in each class and dividing it by the total number of words in that class.

STEP 4: For each document in the test dataset, calculate the posterior probability of each class label using the Naïve Bayes formula

STEP 5: P(class_label | document) = P(class_label)*P(word1 | class_label)*P(word2 | class_label)* ... *P(wordn | class_label)

STEP 6: where word1, word2, ..., wordn are the words in the document and P(word1 | class_label) is the conditional probability of that word given the class label.

STEP 7: Predict the class label with the highest posterior probability for each document in the test dataset.

STEP 8: Return the predicted class labels for the test dataset.

LEARN_NAIVE_BAYES_TEXT (Examples, V)

Example is a set of text documents along with their target values. V is the set of all possibletarget values. This function learns the probability terms P(wk |vj,), describing the probabilitythat a randomly drawn word from a document in class vj will be the English word wk. It also learns the class prior probabilities P(vj).

1. *collect all words, punctuation, and other tokens that occur in Examples*
   - *Vocabulary* ← ***c*** the set of all distinct words and other tokens occurring in any text document from *Examples*

2. *calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms*

   - For each target value v_j in *V* do

     - $docs_j$ ← the subset of documents from *Examples* for which the target value is *vj*

     - $P(v_j)$ ← | $docs_j$ | / |Examples|

- *$Text_j$* ← a single document created by concatenating all members of *$docs_j$*
- *n* ← total number of distinct word positions in *$Text_j$*
- for each word *w_k* in *Vocabulary*
    - *n_k* ← number of times word ***w_k*** occurs in *$Text_j$*
    - *$P(w_k|v_j)$* ← (n_k + 1) / (n + | *Vocabulary*|)

CLASSIFY_NAIVE_BAYES_TEXT (Doc)

Return the estimated target value for the document Doc. a_i denotes the word found in the i^{th}position within Doc.

- *positions* ← all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return *V_{NB},* where

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

Data set:

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

Program:

```
import pandas as pd

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

```
8     0

9     0

10    0

11    1

12    0

13    1

14    0

15    1

16    0

17    1

18    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' 'boss' 'can' 'dance' 'deal'

 'do' 'enemy' 'fun' 'good' 'great' 'have' 'he' 'holiday' 'horrible'

 'house' 'is' 'like' 'locality' 'love' 'my' 'not' 'of' 'place'

 'restaurant' 'sick' 'stay' 'stuff' 'sworn' 'that' 'this' 'tired' 'to'

 'today' 'tomorrow' 'we' 'went' 'what' 'will' 'with']

 Accuracy of the classifier is 1.0

 Confusion matrix

[[2 0]

 [0 3]]

 The value of Precision 1.0

 The value of Recall 1.0
```

Result:

Thus, the Python code to implement Naïve Bayesian classifier model to classify a set of documents was executed and the output was verified successfully.

Ex.No: 7 **Bayesian Network**
Date:

Aim:

To construct a Bayesian network considering medical data to demonstrate the diagnosis of heart patients with standard Heart Disease Data Set using Python code.

Algorithm:

```
Step 1: Constructing a Bayesian Network considering Medical Data.
        Step 1.1: Defining a Structure with nodes and edges.
        Step 1.2: Creation of Conditional Probability Table.
        Step 1.3: Associating Conditional probabilities with the Bayesian Structure.
        Step 1.4: Determining the Local independencies.
        Step 1.5: Inferencing with Bayesian Network.
Step 2: Diagnosis of heart patients using standard Heart Disease Data Set.
        Step 2.1: Importing Heart Disease Data Set and Customizing.
        Step 2.2: Modelling Heart Disease Data.
        Step 2.3: Inferencing with Bayesian Network.
Step 3: Print the results.
```

Procedure:

Introduction:

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable. Bayesian network consists of two major parts: a directed acyclic graph and a set of conditional probability distributions

• The directed acyclic graph is a set of random variables represented by nodes.

• The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

Bayesian Network

• Bayesian Belief networks describe conditional independence among subsets of variables.

→ allows combining prior knowledge about (in)dependencies among variables with observed training data (also called Bayes Nets)

Conditional Independence

• Definition: X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y given the value of Z; that is, if

$$(\forall x_i, y_j, z_k)\ P(X=x_i \mid Y=y_j, Z=z_k) = P(X=x_i \mid Z=z_k)$$

more compactly, we write

$$P(X \mid Y, Z) = P(X \mid Z)$$

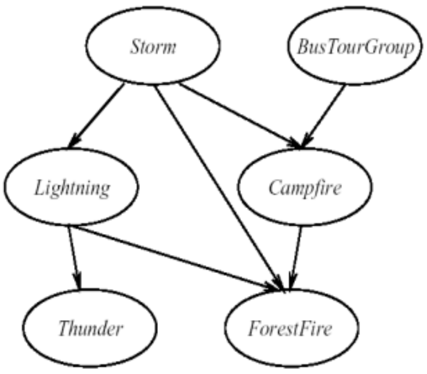
• Example: Thunder is conditionally independent of Rain, given Lightning.

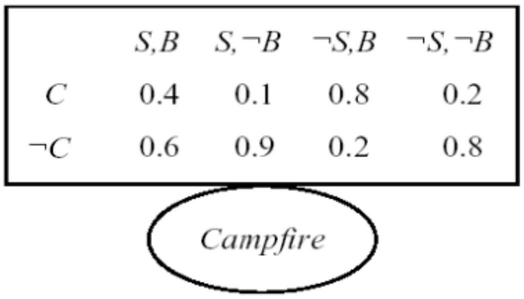
$$P(Thunder \mid Rain, Lightning) = P(Thunder \mid Lightning)$$

• Naive Bayes uses conditional independence to justify

$$P(X, Y \mid Z) = P(X \mid Y, Z)\, P(Y \mid Z) = P(X \mid Z)\, P(Y \mid Z)$$

Bayesian Belief Network





	S,B	$S,\neg B$	$\neg S,B$	$\neg S,\neg B$
C	0.4	0.1	0.8	0.2
$\neg C$	0.6	0.9	0.2	0.8

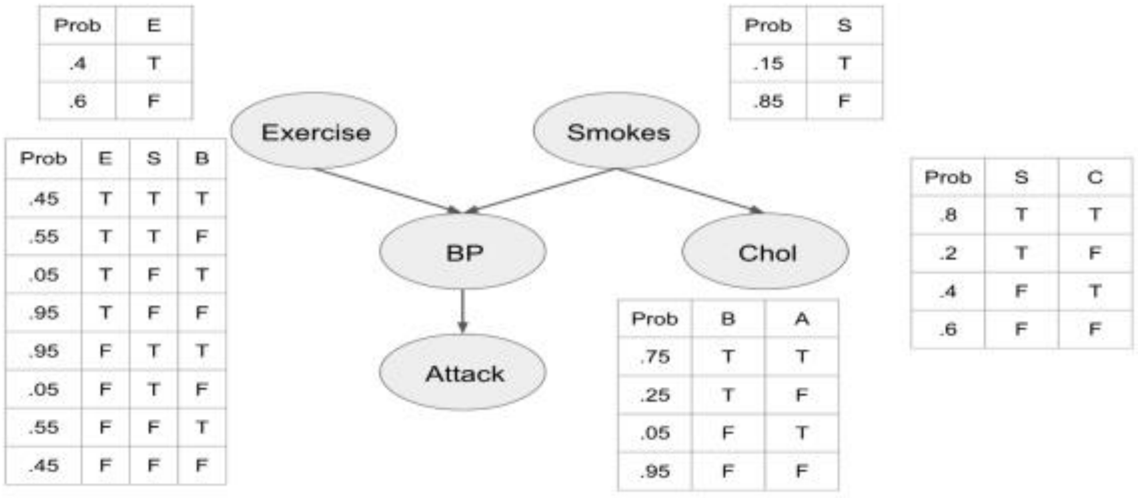
- Represents a set of conditional independence assertions:
  - Each node is asserted to be conditionally independent of its non-descendants, given its immediate predecessors.
  - Directed acyclic graph.
- Represents joint probability distribution over all variables.
  - e.g., ***P*(*Storm*, *BusTourGroup*, . . . , *ForestFire*)**
  - in general,

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

where Parents(Yi) denotes immediate predecessors of Yi in graph

- so, joint distribution is fully defined by graph, plus the $P(y_i | Parents(Y_i))$.

Example 1:



Prob	E
.4	T
.6	F

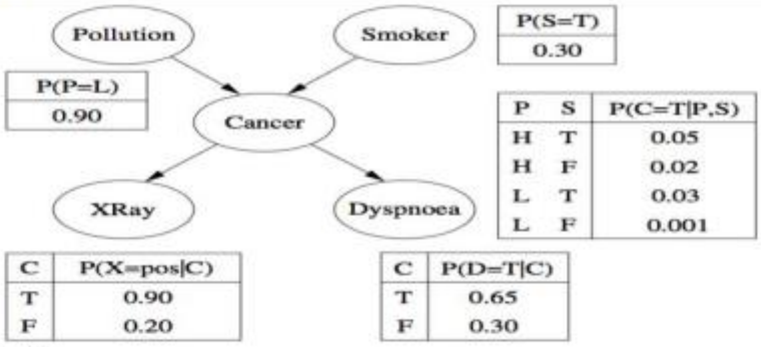
Prob	S
.15	T
.85	F

Prob	E	S	B
.45	T	T	T
.55	T	T	F
.05	T	F	T
.95	T	F	F
.95	F	T	T
.05	F	T	F
.55	F	F	T
.45	F	F	F

Prob	S	C
.8	T	T
.2	T	F
.4	F	T
.6	F	F

Prob	B	A
.75	T	T
.25	T	F
.05	F	T
.95	F	F

Example2 :



P(P=L)
0.90

P(S=T)
0.30

P	S	P(C=T|P,S)
H	T	0.05
H	F	0.02
L	T	0.03
L	F	0.001

C	P(X=pos|C)
T	0.90
F	0.20

C	P(D=T|C)
T	0.65
F	0.30

FIGURE 2.1
A BN for the lung cancer problem.

Data Set:

Title: Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is an integer valued from 0 (no presence) to 4.

Database:	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type

- Value 1: typical angina
- Value 2: atypical angina
- Value 3: non-anginal pain
- Value 4: asymptomatic

4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria.
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment

- Value 1: upsloping
- Value 2: flat
- Value 3: downsloping

12. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
13. Heartdisease: It is integer valued from 0 (no presence) to 4.

Program:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv("C:/Users/Desktop/ML/lab/heart.csv")
heartDisease = heartDisease.replace('?',np.nan)

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

print('\n Attributes and datatypes')
print(heartDisease.dtypes)

model=
BayesianNetwork([('age','heartdisease'),('sex','heartdisease'),('exang'
,'heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('hea
rtdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print('\n 1. Probability of HeartDisease given evidence= restecg')
q1 =
HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'rest
ecg':1})
print(q1)
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  ...  oldpeak  slope  ca  thal
heartdisease

0   63    1   1       145   233  ...      2.3      3   0     6
0

1   67    1   4       160   286  ...      1.5      2   3     3
2

2   67    1   4       120   229  ...      2.6      2   2     7
1

3   37    1   3       130   250  ...      3.5      3   0     3
0
```

```
4   41    0   2       130   204  ...      1.4      1   0     3
0

[5 rows x 14 columns]

 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

INFO:pgmpy: Datatype (N=numerical, C=Categorical Unordered,
O=Categorical Ordered) inferred from data:

 {'age': 'N', 'sex': 'N', 'cp': 'N', 'trestbps': 'N', 'chol': 'N',
'fbs': 'N', 'restecg': 'N', 'thalach': 'N', 'exang': 'N', 'oldpeak':
'N', 'slope': 'N', 'ca': 'C', 'thal': 'C', 'heartdisease': 'N'}

 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 |
+-----------------+---------------------+
```

Result:

Thus, the Python code to construct a Bayesian network to demonstrate the diagnosis of heart patients using a standard Heart Disease Data Set was executed and the output was verified successfully.

Ex.No: 8 **K means Algorithm**
Date:

Aim:

The main aim of this experiment is to explore the k-Means clustering algorithm.

Algorithm:

Step 1: Determine the number of clusters before the algorithm is started. This is called k.
Step 2: Choose k instances randomly. These are initial cluster centers.
Step 3: Compute the mean of the initial clusters and assign the remaining sample to the closest cluster based on Euclidean distance or any other distance measures.
Step 4: Compute the new centroid again considering the newly added samples.
Step 5: Repeat steps 3-4 till the algorithm becomes stable with no more changes in the assignment of instances and clusters.

Program:

```
from sklearn.cluster import KMeans
import numpy as np
X = np.array([[1.713,1.586], [0.180,1.786], [0.353,1.240],
[0.940,1.566], [1.486,0.759],
[1.266,1.106],[1.540,0.419],[0.459,1.799],[0.773,0.186]])
y=np.array([0,1,1,0,1,0,1,1,1])
kmeans = KMeans(n_clusters=3, random_state=0).fit(X,y)
print("The input data is ")
print("VAR1 \t VAR2 \t CLASS")
i=0
for val in X:
    print(val[0],"\t",val[1],"\t",y[i])
    i+=1
print("="*20)
# To get test data from the user
print("The Test data to predict ")
test_data = []
VAR1 = float(input("Enter Value for VAR1 :"))
VAR2 = float(input("Enter Value for VAR2 :"))
test_data.append(VAR1)
test_data.append(VAR2)
print("="*20)
print("The predicted Class is : ",kmeans.predict([test_data]))
```

Output:

```
The input data is
VAR1        VAR2       CLASS
1.713       1.586      0
0.18        1.786      1
0.353       1.24       1
0.94        1.566      0
1.486       0.759      1
1.266       1.106      0
1.54        0.419      1
0.459       1.799      1
0.773       0.186      1
====================
The Test data to predict
Enter Value for VAR1 :1.2
Enter Value for VAR2 :1.75
====================
The predicted Class is :  [1]
```

Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No: 9 **EM algorithm**
Date:

Aim:

To write a Python code to apply EM algorithm to cluster a set of data stored in a .CSV file and use the same data set for clustering using the k-Means algorithm. Compare the results of these two algorithms.

Algorithm:

STEP 1: The very first step is to initialize the parameter values. Further, the system is provided with incomplete observed data with the assumption that data is obtained from a specific model.

STEP 2: This step is known as Expectation or E-Step, which is used to estimate or guess the values of the missing or incomplete data using the observed data. Further, E-step primarily updates the variables.

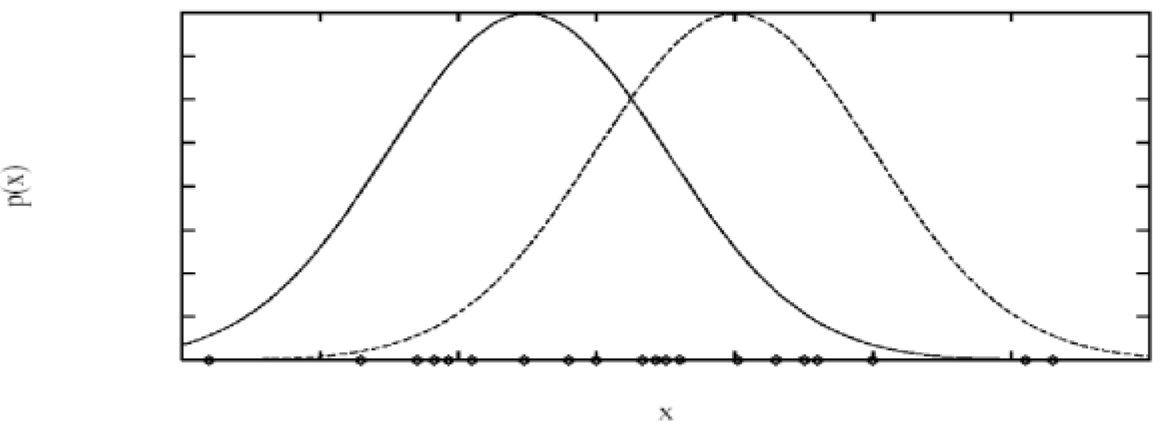
STEP 3: This step is known as Maximization or M-step, where we use complete data obtained from the 2nd step to update the parameter values. Further, M-step primarily updates the hypothesis.

STEP 4: The last step is to check if the values of latent variables are converging or not. If it gets “yes”, then stop the process; else, repeat the process from step 2 until the convergence occurs.

Expectation Maximization (EM) Algorithm

- When to use:
  - Data is only partially observable
  - Unsupervised clustering (target value unobservable)
  - Supervised learning (some instance attributes unobservable)
- Some uses:
  - Train Bayesian Belief Networks
  - Unsupervised clustering (AUTOCLASS)
  - Learning Hidden Markov Models

Generating Data from Mixture of k Gaussians



- Each instance x generated by
  1. Choosing one of the k Gaussians with uniform probability.
  2. Generating an instance at random according to that Gaussian.

EM for Estimating k Means

- Given:
  - Instances from X generated by mixture of k Gaussian distributions
  - Unknown means of the k Gaussians
  - Don’t know which instance xi was generated by which Gaussian
- Determine:

- Maximum likelihood estimates of

- Think of full description of each instance as yi = < xi, zi1, zi2> where
  - zij is 1 if xi generated by jth Gaussian
  - xi observable.
  - zij unobservable.

- **EM Algorithm: Pick random initial $h = \langle \mu_1, \mu_2 \rangle$ then iterate**

  E step: Calculate the expected value E[z_{ij}] of each hidden variable zij, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

$$E[z_{ij}] = \frac{p(x = x_i | \mu = \mu_j)}{\Sigma_{n=1}^2 p(x = x_i | \mu = \mu_n)}$$
$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\Sigma_{n=1}^2 e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}}$$

  M step: Calculate a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, assuming the value taken on by each hidden variable zij is its expected value E[zij] calculated above.

  Replace $h = \langle \mu_1, \mu_2 \rangle$ by $h' = \langle \mu'_1, \mu'_2 \rangle$.

K Means Algorithm

- 1. The sample space is initially partitioned into K clusters and the observations are randomly assigned to the clusters.
- 2. For each sample:
  - Calculate the distance from the observation to the centroid of the cluster.
  - IF the sample is closest to its own cluster THEN leave it ELSE select another cluster.
- 3. Repeat steps 1 and 2 untill no observations are moved from one cluster to another.

Distance functions

Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski $$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

Basic Algorithm of K-means

Algorithm 1 Basic K-means Algorithm.

```
1: Select $K$ points as the initial centroids.
2: repeat
3:     Form $K$ clusters by assigning all points to the closest centroid.
4:     Recompute the centroid of each cluster.
5: until The centroids don't change
```

Details of K-means

1. Initial centroids are often chosen randomly.
   - *Clusters produced vary from one run to another*
2. The centroid is (typically) the mean of the points in the cluster.
3. 'Closeness' is measured by **Euclidean distance**, cosine similarity, correlation, etc.
4. K-means will converge for common similarity measures mentioned above.
5. Most of the convergence happens in the first few iterations.
   - *Often the stopping condition is changed to 'Until relatively few points change clusters'*

Euclidean Distance

$$d(i,j)=\sqrt{|x_{i_1}-x_{j_1}|^2+|x_{i_2}-x_{j_2}|^2+\ldots+|x_{i_p}-x_{j_p}|^2}$$

A simple example: Find the distance between two points, the original and the point (3,4)

$$d_E(O,A)=\sqrt{3^2+4^2}=5$$

Update Centroid

We use the following equation to calculate the n dimensional centroid point amid k n-dimensional points

$$CP(x_1,x_2,\ldots,x_k)=\left(\frac{\sum_{i=1}^{k} x1st_i}{k},\frac{\sum_{i=1}^{k} x2nd_i}{k},\ldots,\frac{\sum_{i=1}^{k} xnth_i}{k}\right)$$

Example: Find the centroid of 3 2D points, (2,4), (5,2) and (8,9)

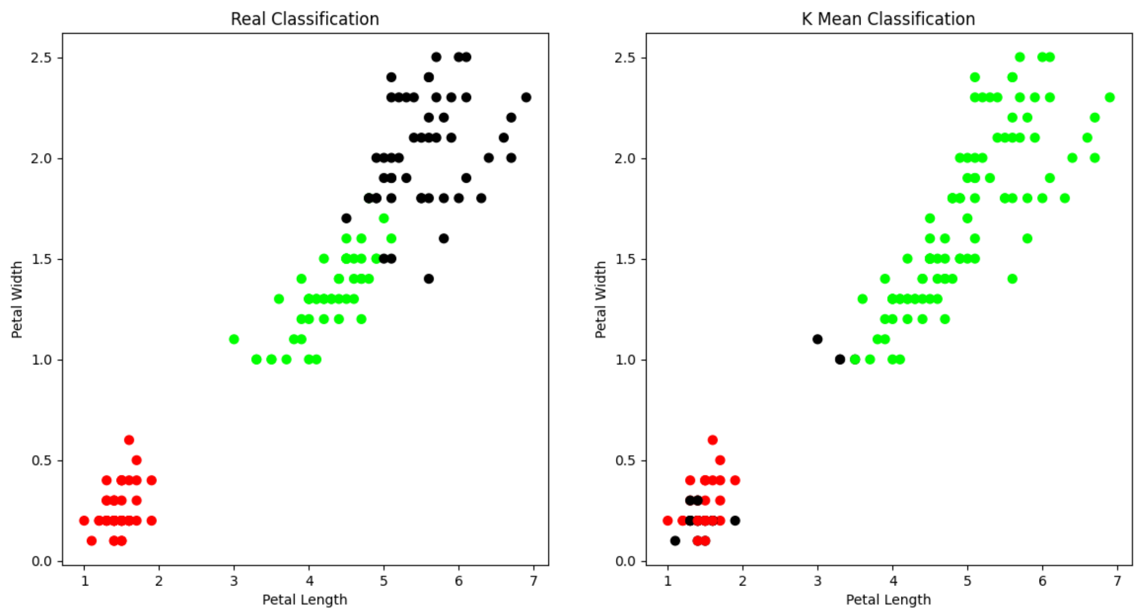
$$CP=\left(\frac{2+5+8}{3},\frac{4+2+9}{3}\right)=(5,5)$$

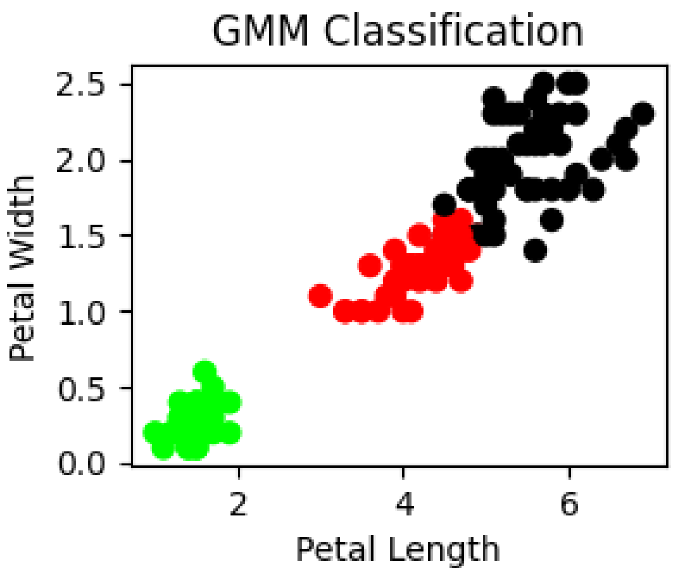
Program:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn import preprocessing
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'])
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')
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')
plt.show()
print('The accuracy score of K-Mean: ', sm.accuracy_score(y,
model.labels_))
print('The Confusion matrix of K-Mean: ', sm.confusion_matrix(y,
model.labels_))
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns=X.columns)
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_gmm = gmm.predict(xs)
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))
plt.show()
```

Output:

```
The accuracy score of K-Mean:  0.52
The Confusion matrix of K-Mean:  [[32  0 18]
 [ 0 46  4]
 [ 0 50  0]]
The accuracy score of EM:  0.3333333333333333
The Confusion matrix of EM:  [[ 0 50  0]
 [45  0  5]
 [ 0  0 50]]
```





Result:

Thus, the Python code to implement to apply EM algorithm to cluster a set of data was executed and the output was verified successfully.

Ex.No: 10 **K-Nearest Neighbor algorithm**
Date:

Aim:

To write a Python code to implement the k-Nearest Neighbour algorithm to classify the iris data set and print both correct and wrong predictions.

Algorithm:

Step 1: Load the data

Step 2: Initialize the value of k

Step 3: To get the predicted class, iterate from 1 to the total number of training data points.

i) Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.

ii) Sort the calculated distances in ascending order based on distance values 3. Get the top k rows from the sorted array

iii) Get the most frequent class of these rows i.e. Get the labels of the selected K entries.

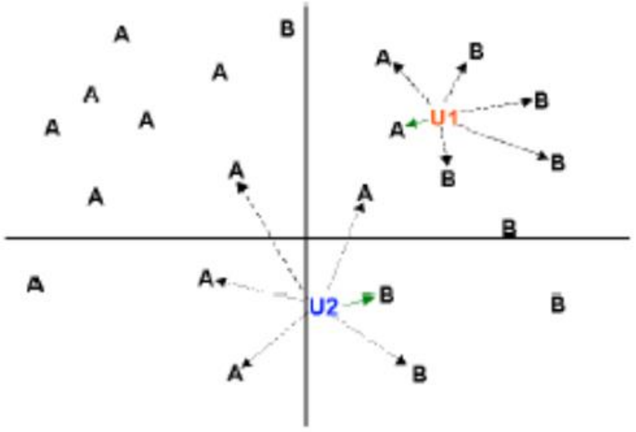
iv) Return the predicted class

- If regression, return the mean of the K labels
- If classification, return the mode of the K labels

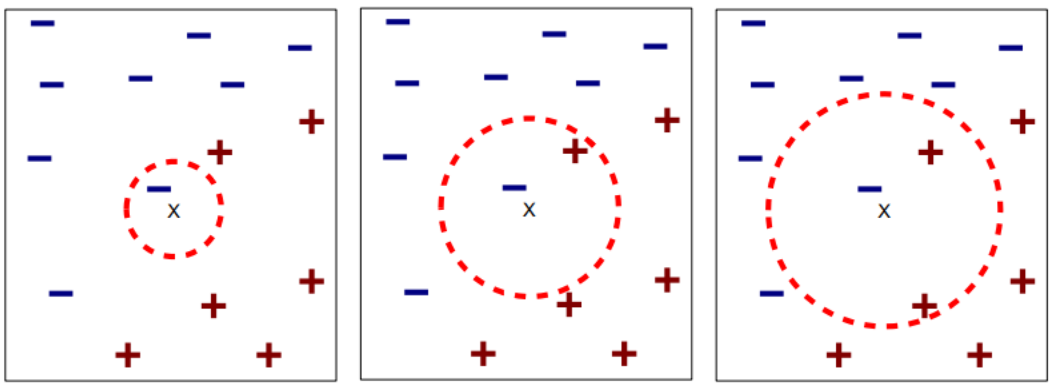
Step 4: End.

K-Nearest-Neighbor:

- Principle: points (documents) that are close in the space belong to the same class.



Definition of Nearest Neighbor:



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Distance Metrics

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r\right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T Q(\boldsymbol{x}-\boldsymbol{y}) = \sum_{j=1}^{m}\left(\sum_{i=1}^{m}(x_i - y_i)q_{ji}\right)(x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m}(\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}} V^{-1}(\boldsymbol{x}-\boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m}(x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m}(x_i - \overline{x_i})^2 \sum_{i=1}^{m}(y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i}\left(\frac{x_i}{size_x} - \frac{y_i}{size_y}\right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)}\sum_{i=1}^{m}\sum_{j=1}^{i-1}\operatorname{sign}(x_i - x_j)\operatorname{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

K-Nearest Neighbor Algorithm:

Training algorithm:

- For each training example (x, f (x)), add the example to the list of training examples

Classification algorithm:

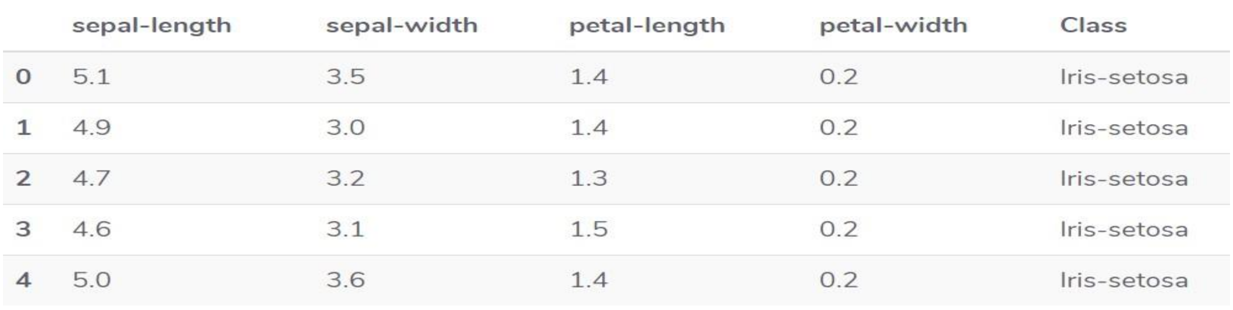
- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

  - Where, $f(x_i)$ function to calculate the mean value of the k nearest training examples.

Data Set:

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes)
Number of Attributes: 4 numeric, predictive attributes and the Class



	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	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]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
 [6.6 2.9 4.6 1.3]
 [5.2 2.7 3.9 1.4]
 [5.  2.  3.5 1. ]
 [5.9 3.  4.2 1.5]
 [6.  2.2 4.  1. ]
 [6.1 2.9 4.7 1.4]
 [5.6 2.9 3.6 1.3]
 [6.7 3.1 4.4 1.4]
 [5.6 3.  4.5 1.5]
 [5.8 2.7 4.1 1. ]
 [6.2 2.2 4.5 1.5]
 [5.6 2.5 3.9 1.1]
 [5.9 3.2 4.8 1.8]
 [6.1 2.8 4.  1.3]
 [6.3 2.5 4.9 1.5]
 [6.1 2.8 4.7 1.2]
 [6.4 2.9 4.3 1.3]
 [6.6 3.  4.4 1.4]
 [6.8 2.8 4.8 1.4]
 [6.7 3.  5.  1.7]
 [6.  2.9 4.5 1.5]
 [5.7 2.6 3.5 1. ]
 [5.5 2.4 3.8 1.1]
 [5.5 2.4 3.7 1. ]
 [5.8 2.7 3.9 1.2]
 [6.  2.7 5.1 1.6]
 [5.4 3.  4.5 1.5]
 [6.  3.4 4.5 1.6]
 [6.7 3.1 4.7 1.5]
 [6.3 2.3 4.4 1.3]
 [5.6 3.  4.1 1.3]
 [5.5 2.5 4.  1.3]
 [5.5 2.6 4.4 1.2]
 [6.1 3.  4.6 1.4]
 [5.8 2.6 4.  1.2]
 [5.  2.3 3.3 1. ]
 [5.6 2.7 4.2 1.3]
 [5.7 3.  4.2 1.2]
 [5.7 2.9 4.2 1.3]
 [6.2 2.9 4.3 1.3]
 [5.1 2.5 3.  1.1]
 [5.7 2.8 4.1 1.3]
 [6.3 3.3 6.  2.5]
 [5.8 2.7 5.1 1.9]
 [7.1 3.  5.9 2.1]
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]
 [7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [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 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
[[21  0  0]
 [ 0  8  3]
 [ 0  0 13]]
Accuracy Metrics
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      0.73      0.84        11
           2       0.81      1.00      0.90        13

    accuracy                           0.93        45
   macro avg       0.94      0.91      0.91        45
weighted avg       0.95      0.93      0.93        45
```

Result:

Thus, the Python code to implement the k-Nearest Neighbor algorithm to classify the iris data set was executed and the output was verified successfully.

Exp.No: 11 **Non-Parametric Locally Weighted Regression algorithm**
Date:

Aim:

To implement the non-parametric Locally Weighted Regression algorithm to fit datapoints. Select the appropriate data set for your experiment and draw graphs.

Algorithm:

Step 1: Read the given data sample to X and the curve (linear or nonlinear) to Y.
Step 2: Set the value for Smoothening parameter or Free parameter say τ.
Step 3: Set the bias /Point of interest set X0, a subset of X.
Step 4: Determine the weight matrix using:

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

Step 5: Determine the value of model term parameter β using:

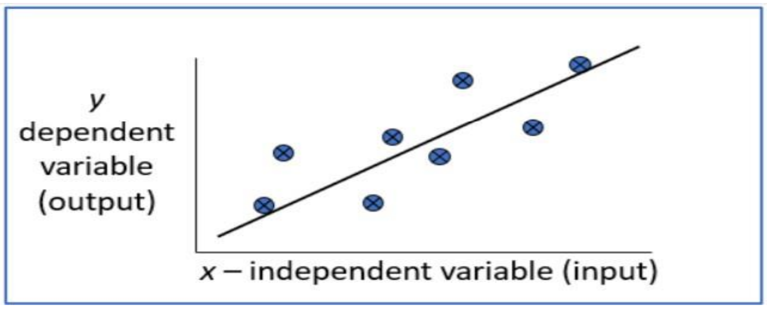
$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

Step 6: Prediction = x_0*β.

Locally Weighted Regression

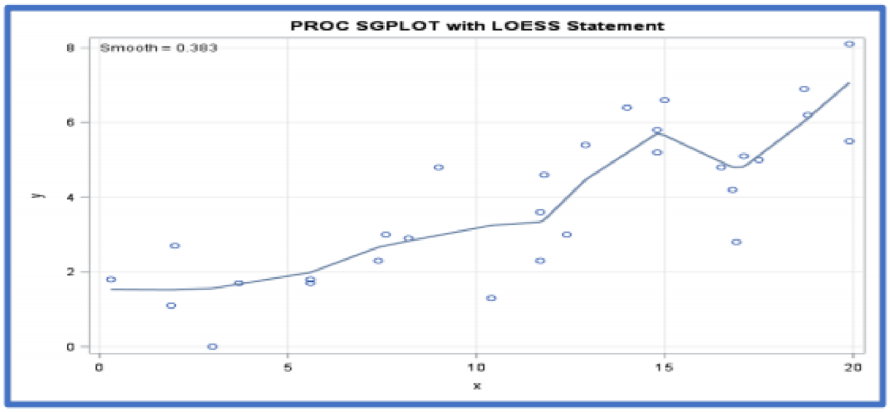
Regression:

- Regression is a technique from statistics that is used to predict values of a desiredtarget quantity when the target quantity is continuous.
- In regression, we seek to identify (or estimate) a continuous variable y associated witha given input vector x.
  - y is called the dependent variable.
  - x is called the independent variable.



Loess/Lowess Regression:

Loess regression is a nonparametric technique that uses locally weighted regression to fit asmooth curve through points in a scatter plot.



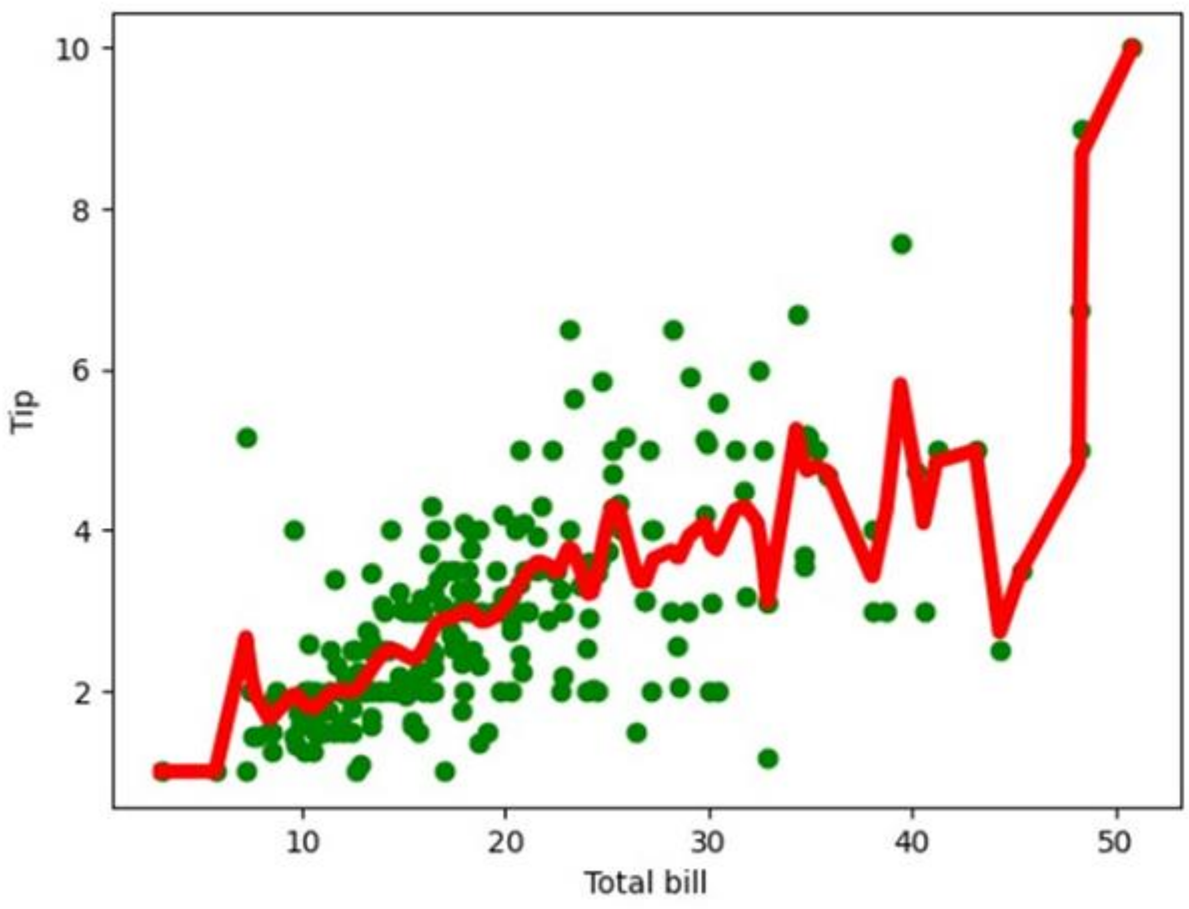
Lowess Algorithm:

Locally weighted regression is a very powerful nonparametric model used in statistical learning. Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors. The weights are given by a kernel function (k or w) which can be chosen arbitrarily.

Program:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
def kernel(point, xmat, k):
    m, n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j, j] = np.exp(diff * diff.T / (-2.0 * k ** 2))
    return weights
def localWeight(point, xmat, ymat, k):
    wei = kernel(point, xmat, k)
    W = (X.T * (wei * X)).I * (X.T * (wei * ymat.T))
    return W
def localWeightRegression(xmat, ymat, k):
    m, n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i] * localWeight(xmat[i], xmat, ymat, k)
    return ypred
data = pd.read_csv("C:/Users /Desktop/ML/lab/tips.csv")
bill = np.array(data.total_bill)
tip = np.array(data.tip)
mbill = np.mat(bill)
mtip = np.mat(tip)
m = np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T, mbill.T))
ypred = localWeightRegression(X, mtip, 0.5)
SortIndex = X[:, 1].argsort(0)
xsort = X[SortIndex][:, 0]
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(bill, tip, color='green')
ax.plot(xsort[:, 1], ypred[SortIndex], color='red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Output:



Result:

Thus, the Python code to implement the non-parametric Locally Weighted Regression algorithm was executed and the output was verified successfully.

Additional Programs

Ex. No:12 **Implementation of Gradient Descent**

Date:

Aim:

To calculate the loss function using MSE and calculate the Gradient of loss function for model parameters.

Algorithm:

```
Step 1: Initialize the data.
Step 2: Estimating weight and bias using gradient descent.
    Step 2.1: Initializing weight, bias, learning rate, and iterations.
    Step 2.2: Estimation of optimal parameters.
        Step 2.2.1: Making predictions.
        Step 2.2.2: Calculating the current cost.
        Step 2.2.3: If the change in cost is less than or equal to stopping_threshold
            we stop the gradient descent.
        Step 2.2.4: Calculate the gradients.
        Step 2.2.5: Update weights and bias.
        Step 2.2.6: Printing the parameters for each 1000th iteration.
    Step 2.3: Visualizing the weights and cost for all iterations.
Step 3: Make predictions using estimated parameters.
Step 4: Plot the regression line.
Step 5: End.
```

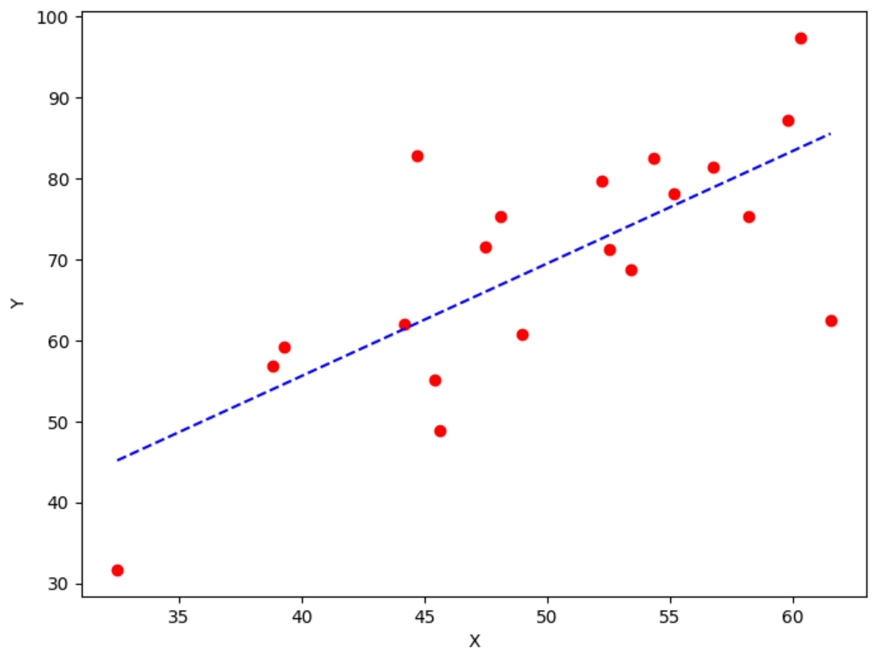
Program:

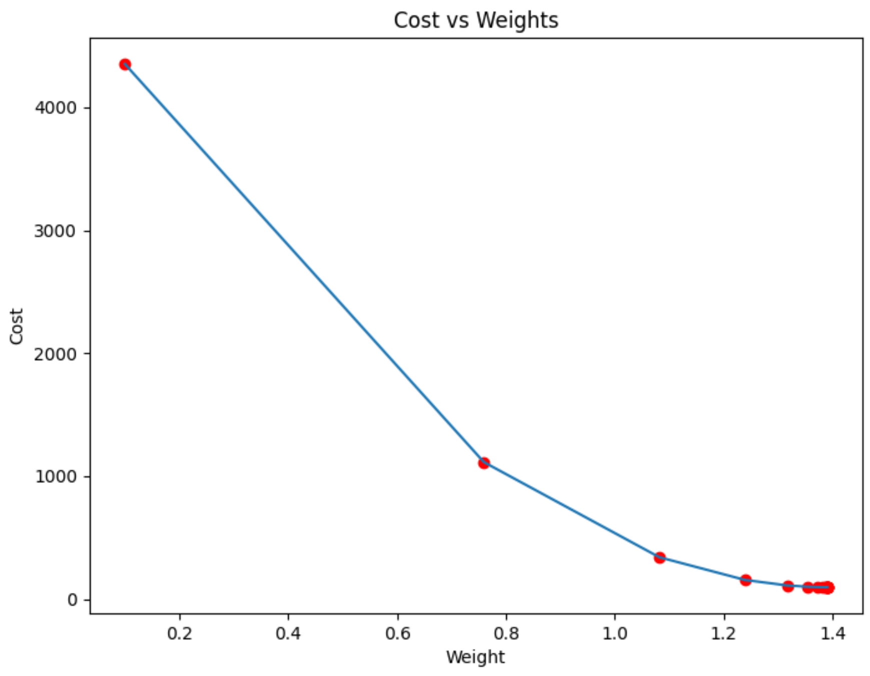
```python
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
def mean_squared_error(y_true, y_predicted):
    # Calculating the loss or cost
    cost = np.sum((y_true - y_predicted) ** 2) / len(y_true)
    return cost
# Gradient Descent Function
# Here iterations, learning_rate, stopping_threshold
# are hyperparameters that can be tuned
def gradient_descent(x, y, iterations=1000, learning_rate=0.0001,
                     stopping_threshold=1e-6):
    # Initializing weight, bias, learning rate and iterations
    current_weight = 0.1
    current_bias = 0.01
    iterations = iterations
    learning_rate = learning_rate
    n = float(len(x))

    costs = []
    weights = []
    previous_cost = None

    # Estimation of optimal parameters
    for i in range(iterations):

        # Making predictions
        y_predicted = (current_weight * x) + current_bias

        # Calculating the current cost
        current_cost = mean_squared_error(y, y_predicted)

        # If the change in cost is less than or equal to
        # stopping_threshold we stop the gradient descent
        if previous_cost and abs(previous_cost - current_cost) <=
stopping_threshold:
            break

        previous_cost = current_cost

        costs.append(current_cost)
        weights.append(current_weight)

        # Calculating the gradients
        weight_derivative = -(2 / n) * sum(x * (y - y_predicted))
        bias_derivative = -(2 / n) * sum(y - y_predicted)

        # Updating weights and bias
        current_weight = current_weight - (learning_rate *
weight_derivative)
        current_bias = current_bias - (learning_rate * bias_derivative)

        # Printing the parameters for each 1000th iteration
        print(f"Iteration {i + 1}: Cost {current_cost}, Weight \
       {current_weight}, Bias {current_bias}")

    # Visualizing the weights and cost at for all iterations
    plt.figure(figsize=(8, 6))
    plt.plot(weights, costs)
    plt.scatter(weights, costs, marker='o', color='red')
    plt.title("Cost vs Weights")
    plt.ylabel("Cost")
    plt.xlabel("Weight")
    plt.show()

    return current_weight, current_bias


def main():
    # Data
    X = np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963,
59.81320787,
                  55.14218841, 52.21179669, 39.29956669, 48.10504169,
52.55001444,
                  45.41973014, 54.35163488, 44.1640495, 58.16847072,
56.72720806,
                  48.95588857, 44.68719623, 60.29732685, 45.61864377,
38.81681754])
    Y = np.array([31.70700585, 68.77759598, 62.5623823, 71.54663223,
87.23092513,
                  78.21151827, 79.64197305, 59.17148932, 75.3312423,
71.30087989,
                  55.16567715, 82.47884676, 62.00892325, 75.39287043,
81.43619216,
                  60.72360244, 82.89250373, 97.37989686, 48.84715332,
56.87721319])

    # Estimating weight and bias using gradient descent
    estimated_weight, estimated_bias = gradient_descent(X, Y,
iterations=2000)
    print(f"Estimated Weight: {estimated_weight}\nEstimated Bias:
{estimated_bias}")

    # Making predictions using estimated parameters
    Y_pred = estimated_weight * X + estimated_bias

    # Plotting the regression line
    plt.figure(figsize=(8, 6))
    plt.scatter(X, Y, marker='o', color='red')
    plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)],
color='blue', markerfacecolor='red',
             markersize=10, linestyle='dashed')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.show()
if __name__ == "__main__":
    main()
```

Output:

```
Iteration 1: Cost 4352.088931274409, Weight        0.7593291142562117,
Bias 0.02288558130709
Iteration 2: Cost 1114.8561474350017, Weight        1.081602958862324,
Bias 0.02918014748569513
Iteration 3: Cost 341.42912086804455, Weight        1.2391274084945083,
Bias 0.03225308846928192
Iteration 4: Cost 156.64495290904443, Weight        1.3161239281746984,
Bias 0.03375132986012604
Iteration 5: Cost 112.49704004742098, Weight        1.3537591652024805,
Bias 0.034479873154934775
Iteration 6: Cost 101.9493925395456, Weight        1.3721549833978113,
Bias 0.03483219539286850 5
Iteration 7: Cost 99.4293893333546, Weight        1.3811467575154601,
Bias 0.03500062439068245
Iteration 8: Cost 98.82731958262897, Weight        1.3855419247507244,
Bias 0.03507916814736111
Iteration 9: Cost 98.68347500997261, Weight        1.3876903144657764,
Bias 0.035113776874486774
Iteration 10: Cost 98.64910780902792, Weight        1.3887405007983562,
Bias 0.035126910596389935
Iteration 11: Cost 98.64089651459352, Weight        1.389253895811451,
Bias 0.03512954755833985
Iteration 12: Cost 98.63893428729509, Weight        1.38950491235671,
Bias 0.035127053821718185
Iteration 13: Cost 98.63846506273883, Weight        1.3896276808137857,
Bias 0.03512205226605122 4
Iteration 14: Cost 98.63835254057648, Weight        1.38968776283053,
Bias 0.0351158249297876 4
```

```
Iteration 15: Cost 98.63832524036214, Weight       1.3897172043139192,
Bias 0.03510899846107016
Iteration 16: Cost 98.63831830104695, Weight       1.389731668997059,
Bias 0.035101879159522745
Iteration 17: Cost 98.63831622628217, Weight       1.389738813163012,
Bias 0.03509461674147458
Estimated Weight: 1.389738813163012
Estimated Bias: 0.03509461674147458
```





Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No:13 **SVM Model**
Date:

Aim:

To build an SVM model using Python.

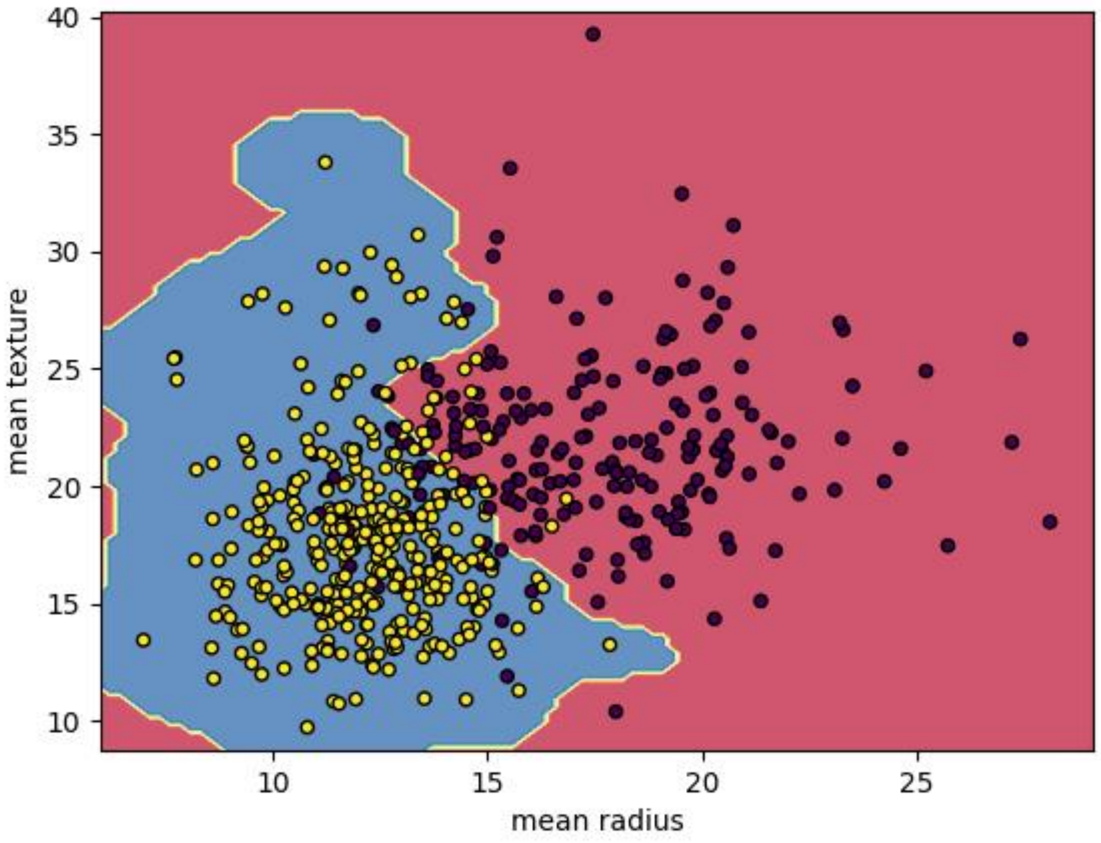
Algorithm:

Step 1: Load the important packages
Step 2: Load the datasets.
Step 3: Build the model.
Step 4: Plot Decision Boundary.
Step 5: Visualize the Scatter plot.

Program:

```
# Load the important packages
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

# Load the datasets
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

# Build the model
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
# Trained the model
svm.fit(X, y)

# Plot Decision Boundary
DecisionBoundaryDisplay.from_estimator(svm,X,response_method="predict",
cmap=plt.cm.Spectral,alpha=0.8,
xlabel=cancer.feature_names[0],ylabel=cancer.feature_names[1])

# Scatter plot
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolors="k")
plt.show()
```

Output:



Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No: 14 **MAXIMUM MARGIN CLASSIFIER**

Date:

Aim:

To build a Maximum Margin Classifier model using Python.

Algorithm:

Step 1: Load the Iris Dataset.
Step 2: Split the data into training and test sets.
Step 3: Create a parameter grid for a grid search.
Step 4: Create the model.
Step 5: Create the grid search object.
Step 6: Fit the grid search object to the training data.
Step 7: Print the best parameters and score.
Step 8: Evaluate the model on the test data.
Step 9: Display the results.
Step 10: Stop.

Program:

```
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
def load_data():
    # Load your data here
    X, y = make_classification(n_samples=1000,
                               n_features=4, random_state=42)
    # Split the data into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42)
    return X_train, X_test, y_train, y_test
# Load the iris dataset
X, y = load_iris(return_X_y=True)
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
# Create a parameter grid for grid search
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
# Create the model
model = LinearSVC(random_state=42)
# Create the grid search object
grid_search = GridSearchCV(model, param_grid, cv=5)
# Fit the grid search object to the training data
grid_search.fit(X_train, y_train)
# Print the best parameters and score
print("Best parameters:", grid_search.best_params_)
print("Best score: {:.2f}".format(grid_search.best_score_))
# Evaluate the model on the test data
accuracy = grid_search.score(X_test, y_test)
print("Test accuracy: {:.2f}".format(accuracy))
```

Output:

```
Best parameters: {'C': 10}
Best score: 0.96
Test accuracy: 1.00
```

Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No: 15 **ENSEMBLE TECHNIQUES**

Date:

Aim:

To implement the ensemble techniques using Python code.

Algorithm:

Step 1: Import utility modules.
Step 2: Import machine learning models for prediction.
Step 3: Load the training data set in the data frame from the train_data.csv file.
Step 4: Get the target data from the data frame.
Step 5: Get the train data from the data frame.
Step 6: Split the training data into training and validation datasets.
Step 7: Initialize all the model objects with default parameters.
Step 8: Train all the models on the training dataset.
Step 9: Predict the output on the validation dataset.
Step 10: Find the final prediction after averaging on the prediction of all 3 models
Step 11: Display the printing of the mean squared error between the real value and the predicted value.

Program:

```
# importing utility modules
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
# importing machine learning models for prediction
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from sklearn.linear_model import LinearRegression
# loading train data set in dataframe from train_data.csv file
df = pd.read_csv(r'C:\Users\Desktop\ML\lab\train_data.csv')
# getting target data from the dataframe
target = df["target"]
# getting train data from the dataframe
train = df.drop("target")
# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split(train, target,
test_size = 0.20)
# initializing all the model objects with default parameters
model_1 = LinearRegression()
model_2 = xgb.XGBRegressor()
model_3 = RandomForestRegressor()
# training all the model on the training dataset
model_1.fit(X_train, y_train)
model_2.fit(X_train, y_train)
model_3.fit(X_train, y_train)
# predicting the output on the validation dataset
pred_1 = model_1.predict(X_test)
pred_2 = model_2.predict(X_test)
pred_3 = model_3.predict(X_test)
# final prediction after averaging on the prediction of all 3 models
pred_final = (pred_1+pred_2+pred_3)/3.0
# printing the mean squared error between real value and predicted
value
print(mean_squared_error(y_test, pred_final))
```

Output:

```
0.05140310393343468
```

Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No:16 **Simple Linear Regression**

Date:

Aim:

To write a Python program for finding linear regression.

Algorithm:

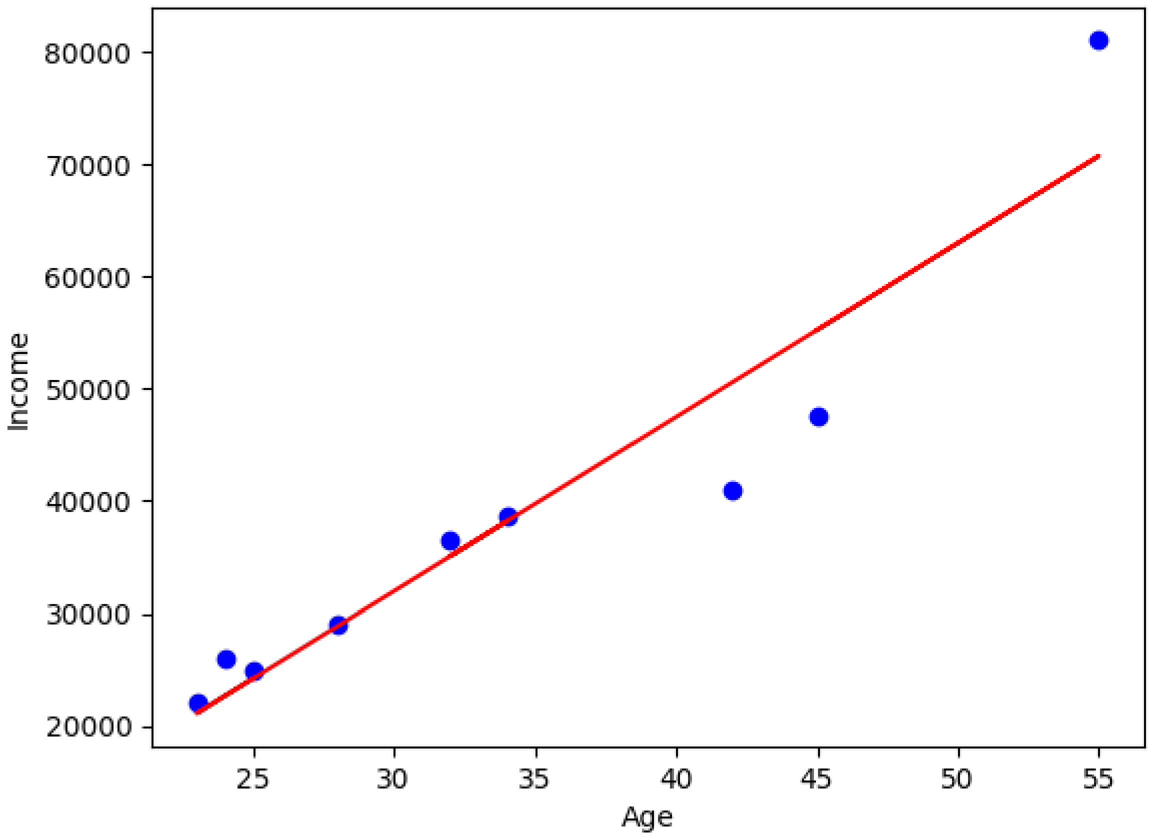
Step 1: Read data from Age_Income.CSV file.
Step 2: Place data into age and income vectors.
Step 3: Find out the number of points.
Step 4: Calculate cross–deviation and deviation about age.
Step 5: Calculate regression coefficients.
Step 6: Display Coefficients.
Step 7: Plot the actual points as scatter plot.
Step 8: Predict response vector.
Step 9: Plot the regression line.
Step 10: Place the labels and display the plots.

Program:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# To read data from Age_Income.csv file
dataFrame = pd.read_csv(r"C:\Users\Desktop\Age_Income.csv")
# To place data in to age and income vectors
age = dataFrame['Age']
income = dataFrame['Income']

# number of points
num = np.size(age)
# To find the mean of age and income vector
mean_age = np.mean(age)
mean_income = np.mean(income)

# calculating cross-deviation and deviation about age
CD_ageincome = np.sum(income*age) - num*mean_income*mean_age
CD_ageage = np.sum(age*age) - num*mean_age*mean_age

# calculating regression coefficients
b1 = CD_ageincome / CD_ageage
b0 = mean_income - b1*mean_age
# to display coefficients
print("Estimated Coefficients :")
print("b0 = ",b0,"\nb1 = ",b1)
# To plot the actual points as scatter plot
plt.scatter(age, income, color = "b",marker = "o")
# TO predict response vector
response_Vec = b0 + b1*age
# To plot the regression line
plt.plot(age, response_Vec, color = "r")
# Placing labels
plt.xlabel('Age')
plt.ylabel('Income')
# To display plot
plt.show()
```

Output:

```
Estimated Coefficients :
b0 =  -14560.45016077166
b1 =  1550.7923748277433
```



Result:

Thus, the program was executed and the output was obtained successfully.

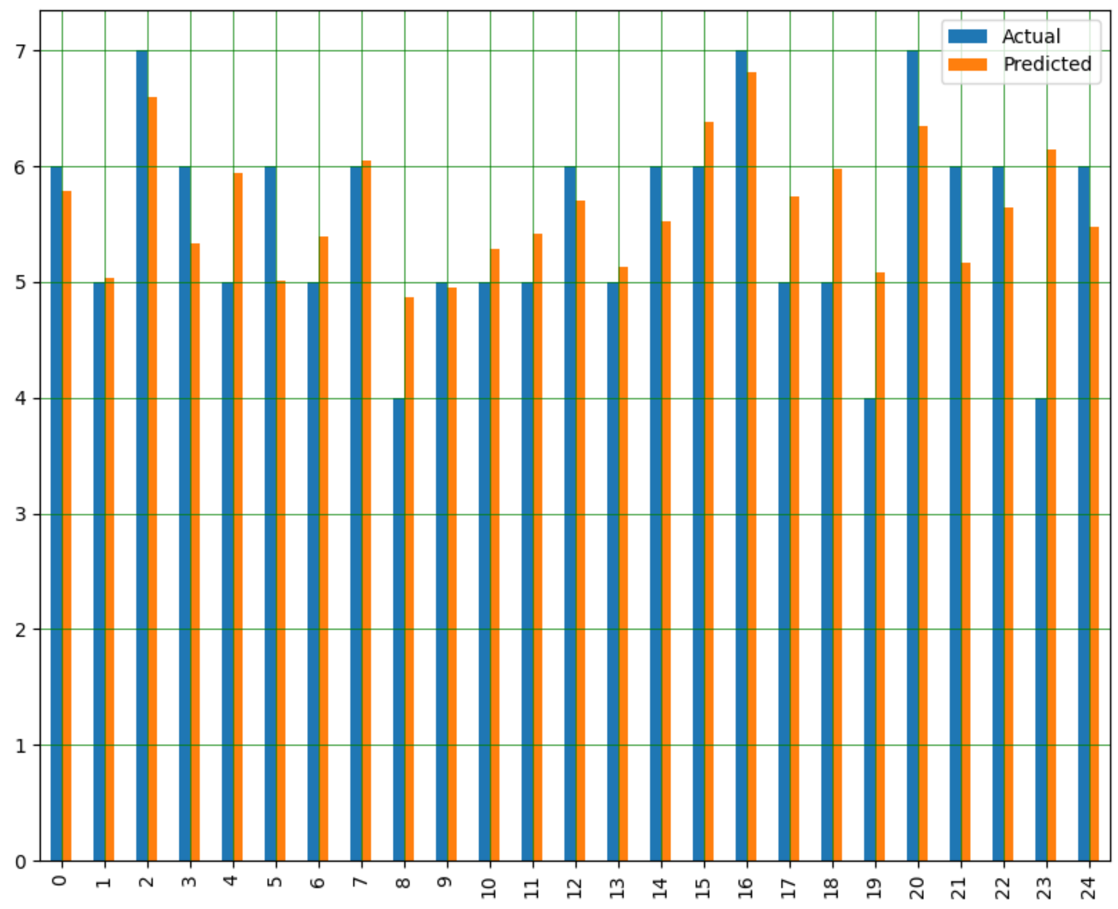
Ex.No:17 **Multiple Linear Regression**
Date:
Aim:
To write a Python program for finding multiple linear regression.

Algorithm:
Step 1: load the dataset.
Step 2: Find out the target variable and feature variables.
Step 3: Check for any null values.
Step 3.1: If present, data should be cleaned before further processing.
Step 4: Create a correlation matrix that measures the linear relationships between the variables.
Step 5: Data is split into training and testing sets.
Step 6: The LinearRegression() function trains the ML model on both the training and test sets.
Step 7: Evaluate the model by computing RMSE score.

Program:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seabornInstance
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
dataset = pd.read_csv(r'C:\Users\Desktop\ML\lab_pgm\winequality-
red.csv')
dataset.isnull().any()
dataset = dataset.fillna(method='ffill')
X = dataset[['fixed acidity', 'volatile acidity', 'citric acid',
'residual sugar', 'chlorides', 'free sulphur dioxide', 'total sulphur
dioxide', 'density', 'pH', 'sulphates', 'alcohol']].values
y = dataset['quality'].values
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df1 = df.head(25)
df1.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-',linewidth='0.5',color='green')
plt.grid(which='minor', linestyle=':',linewidth='0.5',color='black')
plt.show()
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,
y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test,
y_pred))
print('Root Mean Square Error:',
np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

Output:

```
Mean Absolute Error: 0.4696330928661114

Mean Squared Error: 0.38447119782012457

Root Mean Square Error: 0.6200574149384269
```



Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No:18 **Logistic Regression**
Date:
Aim:
To write a Python program to implement logistic regression.

Algorithm:
Step 1: Import required packages, functions, and classes. For logistic regression, we must import LogisticRegression, classification_report(), and confusion_matrix() from scikit-learn.
Step 2: Fetch data, clean it, standardize it, and transform it.
Step 3: Create a classification model.
Step 4: Train (or fit) the model with the existing data.
Step 5: Evaluate the model.
Step 6: If the performance of the model is satisfactory, then use it for predicting new, unseen data.

Program:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
df = pd.read_csv('bank-full.csv',sep=';',header=0)
print(df.head())
print(df.shape)
print(list(df.columns))
df = df.dropna()
data = pd.get_dummies(df, columns =['job', 'marital', 'default',
'housing', 'loan', 'poutcome'])
print(data.head())
print(data.columns)
data.drop(data.columns[[0,1,2,3,4,5,6,7,8,9,10,11,12,16,18,21,24]],axis
=1, inplace=True)
X = data.iloc[:,1:]
Y = data.iloc[:,0]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
random_state=0)
classifier = LogisticRegression(solver='lbfgs', random_state=0)
classifier.fit(X_train, Y_train)
predicted_y = classifier.predict(X_test)
print(predicted_y)
for x in range(len(predicted_y)):
    if(predicted_y[x] == 1):
        print(x, end='\t')
print('Accuracy: {:.2f}'.format(classifier.score(X_test, Y_test)))
```

Output:

```
       job  marital  education default  ...  pdays previous poutcome 
Target
0    management  married   tertiary      no  ...     -1        0  
unknown     no
1    technician   single  secondary      no  ...     -1        0  
unknown     no
2  entrepreneur  married  secondary      no  ...     -1        0  
unknown     no
3   blue-collar  married    unknown      no  ...     -1        0  
unknown     no
4       unknown   single    unknown      no  ...     -1        0  
unknown     no

[5 rows x 16 columns]
(45211, 16)
['job', 'marital', 'education', 'default', 'balance', 'housing', 
'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 
'previous', 'poutcome', 'Target']
   education  balance  ... poutcome_success  poutcome_unknown
0   tertiary     2143  ...            False              True
1  secondary       29  ...            False              True
2  secondary        2  ...            False              True
3    unknown     1506  ...            False              True
4    unknown        1  ...            False              True

[5 rows x 35 columns]
Index(['education', 'balance', 'contact', 'day', 'month', 'duration',
       'campaign', 'pdays', 'previous', 'Target', 'job_admin.',
       'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
       'job_management', 'job_retired', 'job_self-employed', 
'job_services',
       'job_student', 'job_technician', 'job_unemployed', 
'job_unknown',
       'marital_divorced', 'marital_married', 'marital_single', 
'default_no',
       'default_yes', 'housing_no', 'housing_yes', 'loan_no', 
'loan_yes',
       'poutcome_failure', 'poutcome_other', 'poutcome_success',
       'poutcome_unknown'],
      dtype='object')
[False False False ... False False False]
Accuracy: 0.97
```

Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No:19 **Decision Tree Classifier**
Date:
Aim:
To implement the Decision Tree Classifier using Python code.

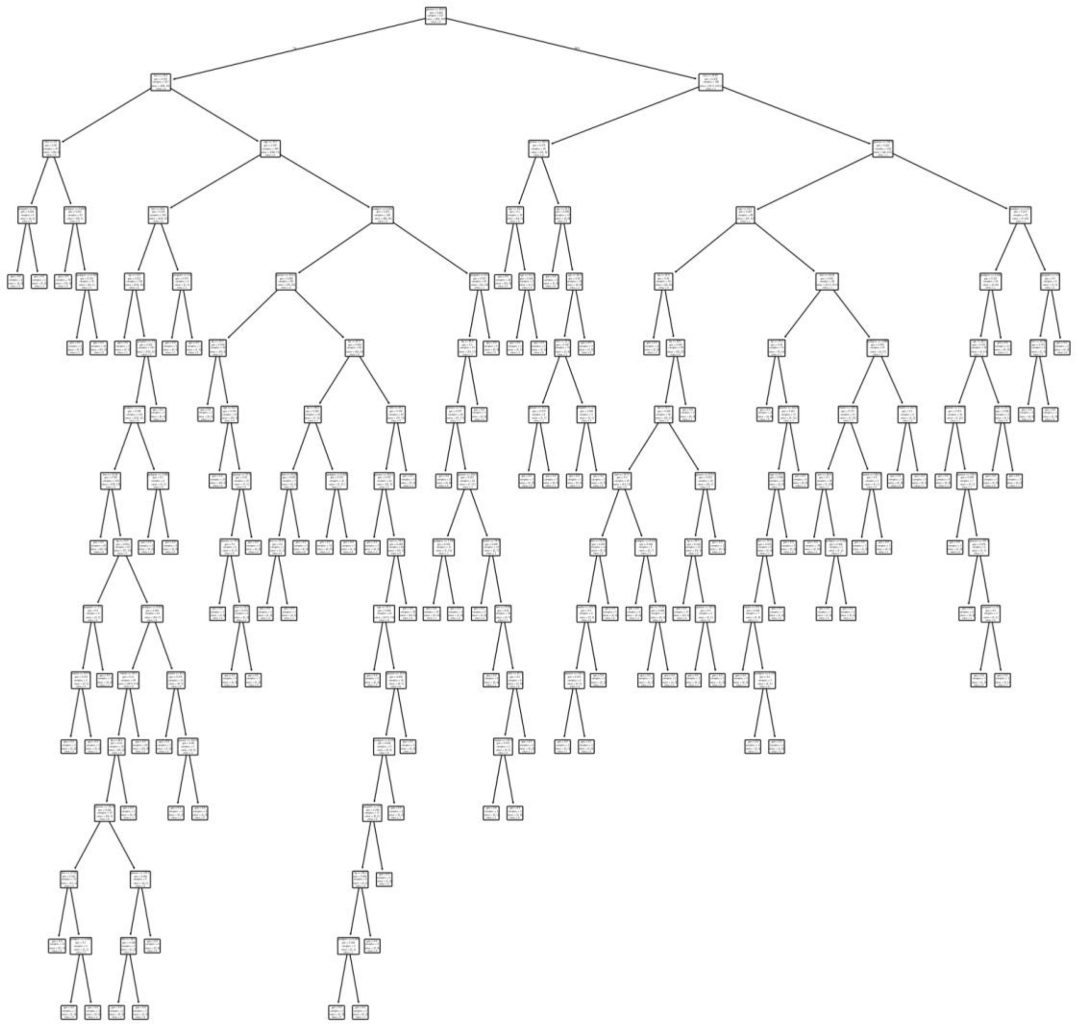
Algorithm:
Step 1: Load Libraries.
Step 2: Load Dataset.
Step 3: Split the dataset into features and target variables.
Step 4: Split the dataset into the training set and test set.
Step 5: Create a Decision Tree Classifier object.
Step 6: Train the Decision Tree classifier.
Step 7: Predict the response for the test dataset.
Step 8: Accuracy of the model shows how often the model is correct.
Step 9: Visualize the graph.
Step 10: Stop.

Program:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import tree

# Define column names
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi',
'pedigree', 'age', 'label']
pima = pd.read_csv(r"C:\Users\Royal traders\Documents\Mohammed
Zaid\diabetes (1).csv", header=None, names=col_names, sep=",",
skiprows=1)
print(pima.head())

# Define features and target
feature_cols = ['pregnant', 'insulin', 'bmi', 'age', 'glucose', 'bp',
'pedigree']
X = pima[feature_cols]
y = pima['label']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=1)

# Initialize and train Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=1)  # Added random_state for
reproducibility
clf.fit(X_train, y_train)

# Predict test data
y_pred = clf.predict(X_test)

# Print accuracy
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

# Plot decision tree
```

```
plt.figure(figsize=(20, 20))
tree.plot_tree(clf, feature_names=feature_cols, class_names=['0', '1'],
rounded=True)
plt.show()
```

Output:

```
     pregnant  glucose  bp  skin  insulin   bmi  pedigree  age  label
0         6      148  72    35        0  33.6     0.627   50      1
1         1       85  66    29        0  26.6     0.351   31      0
2         8      183  64     0        0  23.3     0.672   32      1
3         1       89  66    23       94  28.1     0.167   21      0
4         0      137  40    35      168  43.1     2.288   33      1
Accuracy: 0.6623376623376623
```



Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No:20 **Random Forest Classifier**

Date:

Aim:

To Implement and demonstrate the working of Random Forest classifier and Random Forest regressor using sample data sets. Build the model to classify a test sample.

Algorithm:

Step 1: Import the library 'pandas' to create a Data frame which is a two-dimensional data structure.
Step 2: Import LabelEncoder to normalize labels.
Step 3: Import train_test_split function.
Step 4: Import RandomForestClassifier from sklearn.ensemble
Step 5: Create a list 'data' with the sample dataset.
Step 6: Create pandas dataframe "table" using the structure DataFrame with the given dataset 'data'.
Step 7: Use a value ["CGPA"]=="g9" in the table to select matching row and count the number of columns.
Step 8: Use LabelEncoder() to encode target labels with value between 0 and no_of_classes-1.
Step 9: Then transform non-numerical labels to numerical labels.
Step 10: Use iloc property to select by position.
Step 11: Split the dataset into training dataset and test dataset by using the function train_test_split().
Step 12: Use RandomForestClassifier class. The most important parameter used is n_estimators.
Step 13: Then train the classifier using the function fit().
Step 14: After training, the fitted model can be used to predict a new instance.

Program:

```python
import pandas
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

data = {'CGPA': ['g9', 'l9', 'g9', 'l9', 'g9'],
        'Inter': ['Y', 'N', 'N', 'N', 'Y'],
        'PK': ['++', '==', '==', '==', '=='],
        'CS': ['G', 'G', 'A', 'A', 'G'],
        'Job': ['Y', 'Y', 'N', 'N', 'Y']}

table = pandas.DataFrame(data, columns=["CGPA", "Inter", "PK", "CS",
"Job"])
table.where(table["CGPA"] == "g9").count()
encoder = LabelEncoder()

for i in table:
    table[i] = encoder.fit_transform(table[i])

X = table.iloc[:, 0:4].values
y = table.iloc[:, 4].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=2)
```

```
model = RandomForestClassifier(n_estimators=3)
model.fit(X_train, y_train)

print("\n")
print([0, 1, 1, 1])
if model.predict([[0, 1, 1, 1]]) == 1:
    print("Got JOB")
else:
    print("Didn't get JOB")

print("\n")
print([0, 0, 1, 0])
if model.predict([[0, 0, 1, 0]]) == 1:
    print("Got JOB")
else:
    print("Didn't get JOB")
```

Output:

```
[0, 1, 1, 1]
Got JOB


[0, 0, 1, 0]
Didn't get JOB
```

Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No:21 **Random Forest Regressor**
Date:
Aim:
To implement and demonstrate the working of Random Forest regressor using sample data sets. Build the model to classify a test sample.
Algorithm:
Step 1: Import the required library packages.
Step 2: Load data from a CSV file into a Pandas DataFrame.
Step 3: Use iloc property to select by position.
Step 4: Split the dataset into training dataset and test dataset by using the function train_test_split().
Step 5: Create a new instance of StandardScaler and then fit and transform the scaler to X_train and X_test.
Step 6: Use the RandomForestRegressor model.
Step 7: Train the classifier using the function fit().
Step 8: To make predictions, the predict method of the RandomForestRegressor class is used.
Step 9: Print the Mean Absolute Error and Mean Squared Error.
Step 10: Predict the output for the test sample.
Program:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
dataset = pd.read_csv("C:/Users/Desktop/randomforest.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values
print(" Training Dataset\n")
print("\n", X)
print("\n", y)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
model= RandomForestRegressor(n_estimators=20, random_state=0)
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,
y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test,
y_pred))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print([9.1,85,8])
predicted = model.predict([[9.1,85,8]])
print(predicted)
```

Output:

```
Training Dataset


 [[ 9.2 85.   8. ]
 [ 8.  80.   7. ]
 [ 8.5 81.   8. ]
 [ 6.  45.   5. ]
 [ 6.5 50.   4. ]
 [ 8.2 72.   7. ]
 [ 5.8 38.   5. ]
 [ 8.9 91.   9. ]]

 [95 82 86 60 55 78 48 80]
Mean Absolute Error: 8.850000000000001
Mean Squared Error: 86.16250000000005
Root Mean Squared Error: 9.28237577347524
[9.1, 85, 8]
[91.65]
```

Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No:22 **Perceptron Model**
Date:
Aim:
To implement and demonstrate the perceptron model, a linear binary classifier used for supervised learning.

Algorithm:
```
Step 1: Import numpy, an array-processing package to work with the arrays.
Step 2: Create a Perceptron class to implement a perceptron network.
Step 3: Define the activation function as Step function f(x) .
Step 4: Define the predict function to compute the weighted sum 'z' by multiplying the
inputs with the weights and add the products. Then subtract θ..
Step 5: Define the learning function fit() passing all inputs X, the desired output d, bias
    θ and count.
    Step 5.1: Update the weights for epochs, until the perceptron can correctly classify
    all inputs.
    Step 5.2: Call the predict function, passing the input value x and theta.
    Step 5.3: Calculate error as the difference between the desired output d[i] and the
    predicted output y.
    Step 5.4: Update the weight vector.
Step 6: Define the main function with input array X, desired output array d.
    Step 6.1: Create perceptron object.
    Step 6.2: Call the learning function of the perceptron passing training input X,
    desired output d, theta and count.
    Step 6.3: Print the learned weights for the AND gate which gives the desired output.
```

Program:
```python
import numpy as np
class Perceptron(object):
    def __init__(self, input_size, lr=0.2, epochs=4):
        self.W = np.array([0.3, -0.2])
        self.epochs = epochs
        self.lr = lr
    def activation_fn(self, x):
        return 1 if x >= 0 else 0

    def predict(self, x, theta):
        z = self.W.T.dot(x) - theta
        z = round(z, 2)
        a = self.activation_fn(z)
        return a

    def fit(self, X, d, theta, count):
        for _ in range(self.epochs):

            print("Epoch: ", count)
            count = count + 1
            for i in range(d.shape[0]):
                x = X[i]
                print("input", x, "\t", "Weight:", self.W)
                y = self.predict(x, theta)
                e = d[i] - y
                self.W = self.W + self.lr * e * x
```

```
if __name__ == '__main__':
    X = np.array([
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]
    ])
    d = np.array([0, 0, 0, 1])

    perceptron = Perceptron(input_size=2)
    theta = 0.4
    count = 1
    perceptron.fit(X, d, theta, count)
    print(perceptron.W)
```

Output:

```
Epoch:  1
input [0 0]   Weight: [ 0.3 -0.2]
input [0 1]   Weight: [ 0.3 -0.2]
input [1 0]   Weight: [ 0.3 -0.2]
input [1 1]   Weight: [ 0.3 -0.2]
Epoch:  2
input [0 0]   Weight: [0.5 0. ]
input [0 1]   Weight: [0.5 0. ]
input [1 0]   Weight: [0.5 0. ]
input [1 1]   Weight: [0.3 0. ]
Epoch:  3
input [0 0]   Weight: [0.5 0.2]
input [0 1]   Weight: [0.5 0.2]
input [1 0]   Weight: [0.5 0.2]
input [1 1]   Weight: [0.3 0.2]
Epoch:  4
input [0 0]   Weight: [0.3 0.2]
input [0 1]   Weight: [0.3 0.2]
input [1 0]   Weight: [0.3 0.2]
input [1 1]   Weight: [0.3 0.2]
[0.3 0.2]
```

Result:

Thus, the program was executed and the output was obtained successfully.

Ex.No: 23 **Genetic Algorithm**
Date:
Aim:
To implement and apply a genetic algorithm on the ENJOYSPORT dataset downloaded from Kaggle.
Algorithm:
Step 1: Calculate the fitness of the individuals in a population.
Step 2: Repeat steps 3-6 until the desired number of generations or the stopping criterion is reached.
Step 3: Select the fittest individual.
Step 4: Cross–over the individuals.
Step 5: Mutate the individuals.
Step 6: Calculate the fitness of the individuals again.
Program:
```
import numpy as np
import pandas as pd
data = pd.read_csv('ENJOYSPORT.csv')
print(data)
concepts = np.array(data.iloc[:,0:-1])
print("\n Instances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\n Target Values are:",target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nSpecific Boundary:",specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print("\n Generic Boundary:",general_h)
    for i,h in enumerate(concepts):
        print("\nInstance", i+1 , " is ", h)
        if target[i] == 1:
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
        if target[i] == 0:
            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 Boundary 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")
```

Output:

```
 Example    Sky AirTemp Humidity    Wind Water Forecast EnjoySport
0        1  Sunny    Warm   Normal  Strong  Warm      Same        Yes
1        2  Sunny    Warm     High  Strong  Warm      Same        Yes
2        3  Rainy    Cold     High  Strong  Warm    Change         No
3        4  Sunny    Warm     High  Strong  Cool    Change        Yes

 Instances are:
 [[1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 [2 'Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 [3 'Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 [4 'Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

 Target Values are: ['Yes' 'Yes' 'No' 'Yes']

Specific Boundary: [1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

 Generic Boundary: [['?', '?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?']]

Instance 1  is  [1 'Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Specific Boundary after  1 Instance is  [1 'Sunny' 'Warm' 'Normal'
'Strong' 'Warm' 'Same']
Generic Boundary after  1 Instance is [['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?', '?']]



Instance 2  is  [2 'Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Specific Boundary after  2 Instance is  [1 'Sunny' 'Warm' 'Normal'
'Strong' 'Warm' 'Same']
Generic Boundary after  2 Instance is [['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?', '?']]



Instance 3  is  [3 'Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
Specific Boundary after  3 Instance is  [1 'Sunny' 'Warm' 'Normal'
'Strong' 'Warm' 'Same']
Generic Boundary after  3 Instance is [['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?', '?']]
```

```
Instance 4  is  [4 'Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']
Specific Boundary after  4 Instance is  [1 'Sunny' 'Warm' 'Normal'
'Strong' 'Warm' 'Same']
Generic Boundary after  4 Instance is [['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?', '?']]
```

Result:

Thus, the program was executed and the output was obtained successfully.

Ex.Ndecio: 24 **Bayesian Network (Corona Dataset)**
Date:

Aim:

To construct a Bayesian network considering medical data to demonstrate the diagnosis of corona patients with standard Corona Disease Data Set using Python code.

Algorithm:

```
Step 1: Constructing a Bayesian Network considering Medical Data.
    Step 1.1: Defining a Structure with nodes and edges.
    Step 1.2: Creation of Conditional Probability Table.
    Step 1.3: Associating Conditional probabilities with the Bayesian Structure.
    Step 1.4: Determining the Local independencies.
    Step 1.5: Inferencing with Bayesian Network.
Step 2: Diagnosis of heart patients using standard Corona Disease Data Set.
    Step 2.1: Importing Corona Disease Data Set and Customizing.
    Step 2.2: Modelling Corona Disease Data.
    Step 2.3: Inferencing with Bayesian Network.
Step 3: Print the results.
```

Program:

```
import pandas as pd
from pgmpy.models import DiscreteBayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
# Step 1: Create a synthetic dataset
data = pd.DataFrame([
    ['yes', 'yes', 'yes', 'yes'],
    ['yes', 'yes', 'no', 'yes'],
    ['no', 'yes', 'yes', 'yes'],
    ['no', 'no', 'yes', 'no'],
    ['yes', 'no', 'no', 'no'],
    ['no', 'yes', 'no', 'no'],
    ['yes', 'yes', 'yes', 'yes'],
    ['no', 'no', 'no', 'no'],
    ['yes', 'no', 'yes', 'yes'],
    ['no', 'yes', 'no', 'no'],
], columns=['Fever', 'Cough', 'TravelHistory', 'Corona'])
# Step 2: Define the network structure
model = DiscreteBayesianNetwork([
    ('Fever', 'Corona'),
    ('Cough', 'Corona'),
    ('TravelHistory', 'Corona')
])
# Step 3: Fit the model using Maximum Likelihood Estimation
model.fit(data, estimator=MaximumLikelihoodEstimator)
# Step 4: Inference
inference = VariableElimination(model)
# Step 5: Query - P(Corona | Fever = yes, Cough = yes)
result = inference.query(variables=['Corona'], evidence={'Fever':
'yes', 'Cough': 'yes'})
print(result)
```

Output:
```
INFO:pgmpy: Datatype (N=numerical, C=Categorical Unordered,
O=Categorical Ordered) inferred from data:
 {'Fever': 'C', 'Cough': 'C', 'TravelHistory': 'C', 'Corona': 'C'}
+-------------+---------------+
| Corona      |   phi(Corona) |
+=============+===============+
| Corona(no)  |        0.0000 |
+-------------+---------------+
| Corona(yes) |        1.0000 |
+-------------+---------------+
```

Result:

Thus, the Python code to construct a Bayesian network to demonstrate the diagnosis of corona patients using a standard Corona Disease Data Set was executed and the output was verified successfully.