PROGRAM 1

Aim: Illustrate and Demonstrate the working model and principle of Find-S algorithm.
Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

ALGORITHM:

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

PROGRAM:

```
import csv
a=[]
with open('enjoysport.csv','r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)
    for i in range(0,len(a)):
        print(a[i])
print("\nThe total number of training instances are:",len(a))
num_attribute=len(a[0])-1
print("\nThe 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("\nThe hypothesis for the training instance {}
is:\n".format(i+1),hypothesis)
print("\n The maximally specific hypothesis for the training instance
is")
print(hypothesis)
```

TRAINING DATA:

	A	B	C	D	E	F	G
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
5							

OUTPUT:

['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: 4

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

The hypothesis for the training instance 1 is:
['SUNNY', 'WARM', 'NORMAL', 'STRONG', 'WARM', 'SAME']

The hypothesis for the training instance 2 is:
['SUNNY', 'WARM', '?', 'STRONG', 'WARM', 'SAME']

The hypothesis for the training instance 3 is:
['SUNNY', 'WARM', '?', 'STRONG', 'WARM', 'SAME']

The hypothesis for the training instance 4 is:
['SUNNY', 'WARM', '?', 'STRONG', '?', '?']

The maximally specific hypothesis for the training instance is
['SUNNY', 'WARM', '?', 'STRONG', '?', '?']

PROGRAM 2

Aim: Demonstrate the working model and principle of candidate elimination algorithm.

Program: 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:

`Initialize G to the set of maximally general hypotheses in H`

`Initialize S to the set of maximally specific hypotheses in H`

`For each training example d, do`

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

PROGRAM :

```
import numpy as np
imporst pandas as pd
data = pd.DataFrame(data=pd.read_csv('enjoysport2.csv'))
concepts = np.array(data.iloc[:,0:-1])
print("Initial Concepts:")
print(concepts)
print("\nInitial Targets:")
target = np.array(data.iloc[:,-1])
print(target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization 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):
        if target[i] == "YES":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
        if target[i] == "NO":
            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("\nSteps of Candidate Elimination Algorithm",i+1,":")
        print(specific_h)
        print(general_h)
    indices = [i for i, val in enumerate(general_h) if val == ['?',
'?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("\nFinal Specific_h:", s_final, sep="\n")
print("\nFinal General_h:", g_final, sep="\n")
```

TRAINING DATA:

	A	B	C	D	E	F	G
1	SKY	AIRTEMP	HUMIDITY	WIND	WATER	FORECAST	ENJOYSPORT
2	SUNNY	WARM	NORMAL	STRONG	WARM	SAME	YES
3	SUNNY	WARM	HIGH	STRONG	WARM	SAME	YES
4	RAINY	COLD	HIGH	STRONG	WARM	CHANGE	NO
5	SUNNY	WARM	HIGH	STRONG	COOL	CHANGE	YES
6							

OUTPUT:

Initial Concepts:

[['SUNNY' 'WARM' 'NORMAL' 'STRONG' 'WARM' 'SAME']
 ['SUNNY' 'WARM' 'HIGH' 'STRONG' 'WARM' 'SAME']
 ['RAINY' 'COLD' 'HIGH' 'STRONG' 'WARM' 'CHANGE']
 ['SUNNY' 'WARM' 'HIGH' 'STRONG' 'COOL' 'CHANGE']]

Initial Targets:

['YES' 'YES' 'NO' 'YES']

Initialization of specific_h and general_h:

['SUNNY' 'WARM' 'NORMAL' 'STRONG' 'WARM' 'SAME']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 1 :

['SUNNY' 'WARM' 'NORMAL' 'STRONG' 'WARM' 'SAME']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2 :

['SUNNY' 'WARM' '?' 'STRONG' 'WARM' 'SAME']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3 :

['SUNNY' 'WARM' '?' 'STRONG' 'WARM' 'SAME']
[['SUNNY', '?', '?', '?', '?', '?'], ['?', 'WARM', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'SAME']]

Steps of Candidate Elimination Algorithm 4 :

['SUNNY' 'WARM' '?' 'STRONG' '?' '?']
[['SUNNY', '?', '?', '?', '?', '?'], ['?', 'WARM', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:

['SUNNY' 'WARM' '?' 'STRONG' '?' '?']

Final General_h:

[['SUNNY', '?', '?', '?', '?', '?'], ['?', 'WARM', '?', '?', '?', '?']]

PROGRAM 3

Aim: To construct the Decision tree using the training data sets under supervised learning concept.

Program: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

ALGORITHM:

ID3(Examples, Target_attribute, Attributes)

- Create a Root node for the tree
- If all Examples are positive
    - Return the single-node tree Root, with label = +
- If all Examples are negative
    - Return the single-node tree Root, with label = -
- If Attributes is empty
    - Return the single-node tree Root,
- with label = most common value of Target_attribute in Examples
- Otherwise Begin
    - A← the attribute from Attributes that best* classifies Examples
    - The decision attribute for Root ← A
    - For each possible value, vi, of A,
        - Add a new tree branch below Root, corresponding to the test A = vi
        - Let Examples vi, be the subset of Examples that have value vi for A
        - If Examples vi, is empty
            - Then below this new branch add a leaf node with
            - label = most common value of Target_attribute in Examples
        - Else
            - below this new branch
            - add the subtree
- End
- Return Root

PROGRAMS:

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

data = pd.read_csv("playtennis.csv")
features = [feat for feat in data]
features.remove("Playtennis")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["Playtennis"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        sub_e =entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    return gain
def ID3(examples, attrs):
    root = Node()
    max_gain= 0
    max_feat = ""
    for feature in attrs:
```

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

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t",end="")
    print(root.value, end="")
    if root.isLeaf:
        print("->",root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
root = ID3(data, features)
printTree(root)
```

TRAINING DATA:

	A	B	C	D	E
1	outlook	temp	humidity	wind	Playtennis
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no
16					

OUTPUT:

```
outlook
        overcast-> ['yes']

        rain
                wind
                        strong-> ['no']

                        weak-> ['yes']

        sunny
                humidity
                        high-> ['no']

                        normal-> ['yes']
```

PROGRAM 4:

Aim: To understand the working principle of Artificial Neural network with feed forward and feed backward principle.

Program: 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 (x,t), where (x) is the vector of network input values, (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 as 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 (x, E), in training examples. Do
        - Propagate the input forward through the network:
            - Input the instance x, to the network and compute the output o_u, of every unit u in the network.
        - Propagate the errors backward through the network:
            - For each network output unit k, calculate its error term δ_k.

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

            - For each hidden unit k, calculate its error term δ_h.

              $$\delta_h \leftarrow O_h(1-O_h) \sum_{k \epsilon outputs} W_{h,k}\delta_k$$

            - Update each network weight w_{ji}.

              $$W_{ji} \leftarrow W_{ji} + \triangle W_{ji}$$

              Where

              $$\triangle W_{ji} = \eta \delta_j x_{i,j}$$

PROGRAM:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0)
def sigmoid (x):
    return 1/(1 + np.exp(-x))
def derivatives_sigmoid(x) :
    return x * (1 - x)
epoch=5
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act =sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp=outinp1+bout
    output = sigmoid(outinp)
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
    wh+= X.T.dot(d_hiddenlayer) *lr
    print("-----------Epoch-", i+1,"Starts----------------------")
    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
    print ("-----------Epoch-", i+1, "Ends \n")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n",output)
```

OUTPUT:

```
-----------Epoch- 1 Starts----------------------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.69307797]
 [0.67985154]
 [0.69294354]]
-----------Epoch- 1 Ends

-----------Epoch- 2 Starts----------------------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.99999742]
 [0.99999142]
 [0.99999736]]
-----------Epoch- 2 Ends
```

```
-----------Epoch- 3 Starts----------------------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.99999742]
 [0.99999143]
 [0.99999737]]
-----------Epoch- 3 Ends

-----------Epoch- 4 Starts----------------------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.99999742]
 [0.99999143]
 [0.99999737]]
-----------Epoch- 4 Ends
```

```
-----------Epoch- 5 Starts----------------------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.99999743]
 [0.99999143]
 [0.99999737]]
-----------Epoch- 5 Ends

Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.99999743]
 [0.99999143]
 [0.99999737]]
```

PROGRAM 5

Aim: Demonstrate the text classifier using Naïve bayes classifier algorithm.
Program: Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

PROGRAM:

```
import numpy as np
import random
import csv
import pdb
def read_data(filename):
    with open(filename,'r')as csvfile:
        datareader = csv.reader(csvfile)
        metadata=next(datareader)
        traindata=[]
        for row in datareader:
            traindata.append(row)
        return (metadata,traindata)
def splitdataset(dataset,splitratio):
    trainsize=int(len(dataset)*splitratio)
    trainset=[]
    testset=list(dataset)
    i=0
    while len(trainset)<trainsize:
        trainset.append(testset.pop(i))
    return [trainset,testset]
def classifydata(data,test):
    total_size=data.shape[0]
    print("\n")
    print("training data size=",total_size)
    print("test data size=",test.shape[0])
    countyes=0
    countno=0
    probyes=0
    probno=0
    print("\n")
    print("target count probability")
    for x in range(data.shape[0]):
        if data[x,data.shape[1]-1]=='yes':
            countyes=countyes+1
        if data[x,data.shape[1]-1]=='no':
            countno=countno+1
    probyes=countyes/total_size
    probno=countno/total_size
```

```
    print("yes","\t",countyes,"\t",probyes)
    print("no","\t",countno,"\t",probno)
    prob0=np.zeros((test.shape[1]-1))
    prob1=np.zeros((test.shape[1]-1))
    accuracy=0
    print("\n")
    print("instance prediction target")
    for t in range(test.shape[0]):
        for k in range(test.shape[1]-1):
            count1=count0=0
            for j in range(data.shape[0]):
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='no':
                    count0=count0+1
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='yes':
                    count1=count1+1
            prob0[k]=count0/countno
            prob1[k]=count1/countyes
        probNo=probno
        probYes=probyes
        for i in range(test.shape[1]-1):
            probNo=probNo*prob0[i]
            probYes=probYes*prob1[i]
        if probNo>probYes:
            predict='no'
        else:
            predict='yes'
        print(t+1,"\t",predict,"\t",test[t,test.shape[1]-1])
        if predict==test[t,test.shape[1]-1]:
            accuracy+=1
    final_accuracy=(accuracy/test.shape[0])*100
    print("accuracy",final_accuracy,"%")
    return
metadata,traindata=read_data("5.csv")
print("attribute names of the training data are:", metadata)
splitratio=0.6
trainingset,testset=splitdataset(traindata,splitratio)
training=np.array(trainingset)
print("\n training data set are")
for x in trainingset:
    print(x)
testing=np.array(testset)
print("\n the test data set are:")
for x in testing:
    print(x)
classifydata(training, testing)
```

Training Data:

	A	B	C	D	E	F
1	Day	Outlook	Temperature	Humidity	Wind	PlayTennis
2	D1	Sunny	Hot	High	Weak	no
3	D2	Sunny	Hot	High	Strong	no
4	D3	Overcast	Hot	High	Weak	yes
5	D4	Rain	Mild	High	Weak	yes
6	D5	Rain	Cool	Normal	Weak	yes
7	D6	Rain	Cool	Normal	Strong	no
8	D7	Overcast	Cool	Normal	Strong	yes
9	D8	Sunny	Mild	High	Weak	no
10	D9	Sunny	Cool	Normal	Weak	yes
11	D10	Rain	Mild	Normal	Weak	yes
12	D11	Sunny	Mild	Normal	Strong	yes
13	D12	Overcast	Mild	High	Strong	yes
14	D13	Overcast	Hot	Normal	Weak	yes
15	D14	Rain	Mild	High	Strong	no
16						

OUTPUT:

attribute names of the training data are: ['Day', 'Outlook', 'Temperature', 'Humidity', 'Wind', 'PlayTennis']

training data set are
['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']

the test data set are:
['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']

training data size= 8
test data size= 6

target	count	probability
yes	4	0.5
no	4	0.5

instance	prediction	target
1	yes	yes
2	yes	yes
3	yes	yes
4	yes	yes
5	yes	yes
6	yes	no

accuracy 83.33333333333334 %

PROGRAM 6

Aim: Demonstrate and Analyse the results sets obtained from Bayesian belief network Principle.

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

PROGRAM:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
from sklearn.preprocessing import LabelEncoder

data=pd.read_csv("6.csv",
names=['age','Gender','Family','diet','Lifestyle','cholestrol','heartdi
sease'])
heartDisease=pd.DataFrame(data)
heartDisease.columns
lb = LabelEncoder()
for col in heartDisease.columns:
    heartDisease[col] = lb.fit_transform(heartDisease[col])
print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
model=
BayesianModel([('age','heartdisease'),('Gender','heartdisease'),('Famil
y','heartdisease'),('diet','cholestrol'),('Lifestyle','diet'),('heartdi
sease','cholestrol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1,
MiddleAged:2, Youth:3, Teen:4 }')
print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')
print('For diet Enter { High:0, Medium:1 }')
```

```
print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2,
Sedentary:3 }')
print('For cholesterol Enter { High:0, BorderLine:1, Normal:2 }')

print('\n 1. Probability of HeartDisease given evidence= age')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'a
ge':int(input("Enter age"))})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cholestrol ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'c
holestrol':int(input("Enter Cholestrol"))})
print(q2)
```

TRAINING DATA:

Note: Download Data set from Kaggle.

Sample:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	
2	63	1	0	145	233	1	2	150	0	2.3	2	0	2	0	
3	67	1	3	160	286	0	2	108	1	1.5	1	3	1	1	
4	67	1	3	120	229	0	2	129	1	2.6	1	2	3	1	
5	37	1	2	130	250	0	0	187	0	3.5	2	0	1	0	
6	41	0	1	130	204	0	2	172	0	1.4	0	0	1	0	
7	56	1	1	120	236	0	0	178	0	0.8	0	0	1	0	
8	62	0	3	140	268	0	2	160	0	3.6	2	2	1	1	
9	57	0	3	120	354	0	0	163	1	0.6	0	0	1	0	
10	63	1	3	130	254	0	2	147	0	1.4	1	1	3	1	
11	53	1	3	140	203	1	2	155	1	3.1	2	0	3	1	
12	57	1	3	140	192	0	0	148	0	0.4	1	0	2	0	
13	56	0	1	140	294	0	2	153	0	1.3	1	0	1	0	

OUTPUT:

```
Sample instances from the dataset are given below
                                      age  Gender  Family  diet  Lifestyle  cholestrol  heartdisease
age sex cp trestbps chol fbs restecg  91       2      40     3          4           3             2
63  1   0  145      233  1   2        42       0      22     2          0           1             0
67  1   3  160      286  0   2         3       1      15     1          3           0             1
           120      229  0   2        22       1      25     1          2           2             1
37  1   2  130      250  0   0        77       0      32     2          0           0             0

 Attributes and datatypes
age             int32
Gender          int32
Family          int32
diet            int32
Lifestyle       int32
cholestrol      int32
heartdisease    int32
dtype: object
```

```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:
For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }
For Gender Enter { Male:0, Female:1 }
For Family History Enter { yes:1, No:0 }
For diet Enter { High:0, Medium:1 }
For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }
For cholesterol Enter { High:0, BorderLine:1, Normal:2 }

 1. Probability of HeartDisease given evidence= age
Enter age50
```

```
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.2855 |
+-----------------+---------------------+
| heartdisease(1) |              0.4820 |
+-----------------+---------------------+
| heartdisease(2) |              0.2325 |
+-----------------+---------------------+

 2. Probability of HeartDisease given evidence= cholestrol
Enter Cholestrol3
```

```
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.0051 |
+-----------------+---------------------+
| heartdisease(1) |              0.0041 |
+-----------------+---------------------+
| heartdisease(2) |              0.9909 |
+-----------------+---------------------+
```

PROGRAM 7

Aim: Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept.

Program: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using the k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API in the program.

ALGORITHM:

EM Algorithm:

Step 1: Given a set of incomplete data, consider a set of starting parameters.

Step 2: Expectation step (E - step): Using the observed available data of the dataset, estimate (guess) the values of the missing data.

Step 3: Maximization step (M - step): Complete data generated after the expectation (E) step is used in order to update the parameters.

Step 4: Repeat step 2 and step 3 until convergence.

K- Means Clustering:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be different from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which mean reassign each data point to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

TRAINING DATA:

Note: Download Data from Kaggle

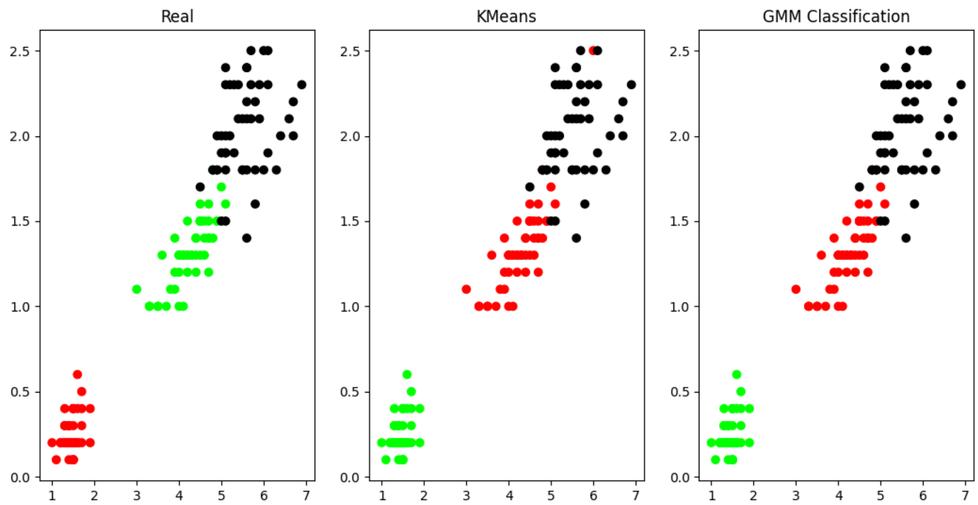
	A	B	C	D	E	F	G
1	Id	SepalLeng	SepalWidt	PetalLengt	PetalWidtl	Species	
2	1	5.1	3.5	1.4	0.2	Iris-setosa	
3	2	4.9	3	1.4	0.2	Iris-setosa	
4	3	4.7	3.2	1.3	0.2	Iris-setosa	
5	4	4.6	3.1	1.5	0.2	Iris-setosa	
6	5	5	3.6	1.4	0.2	Iris-setosa	
7	6	5.4	3.9	1.7	0.4	Iris-setosa	
8	7	4.6	3.4	1.4	0.3	Iris-setosa	
9	8	5	3.4	1.5	0.2	Iris-setosa	
10	9	4.4	2.9	1.4	0.2	Iris-setosa	

PROGRAM:

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset = pd.read_csv("Iris.csv")
print(dataset.columns)
X = dataset.iloc[:, :-1]
y = dataset['Species']
label = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
y = [label[c] for c in y]
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
plt.subplot(1, 3, 1)
plt.title('Real')
plt.scatter(X['PetalLengthCm'], X['PetalWidthCm'], c=colormap[y])
model = KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1, 3, 2)
plt.title('KMeans')
plt.scatter(X['PetalLengthCm'], X['PetalWidthCm'],
c=colormap[model.labels_])
print("The accuracy score of K-Mean: ", metrics.accuracy_score(y,
model.labels_))
print("The Confusion matrix of K-Mean:\n", metrics.confusion_matrix(y,
model.labels_))
gmm = GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm = gmm.predict(X)
plt.subplot(1, 3, 3)
plt.title('GMM Classification')
plt.scatter(X['PetalLengthCm'], X['PetalWidthCm'],
c=colormap[y_cluster_gmm])
print("The accuracy score of EM: ", metrics.accuracy_score(y,
y_cluster_gmm))
print("The Confusion matrix of EM:\n", metrics.confusion_matrix(y,
y_cluster_gmm))
plt.show()
```

OUTPUT:

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
The accuracy score of K-Mean:  0.3333333333333333
The Confusion matrix of K-Mean:
 [[ 0 50  0]
 [49  1  0]
 [ 1  0 49]]
The accuracy score of EM:  0.3333333333333333
The Confusion matrix of EM:
 [[ 0 50  0]
 [50  0  0]
 [ 0  0 50]]
```



PROGRAM 8

Aim: Demonstrate and analyse the results of classification based on KNN Algorithm.
Program: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

ALGORITHM:

K-Nearest Neighbor

Training algorithm:

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

Classification algorithm:

- Given a query instance xq to be classified,
- Let x1 ...xk denote the k instances from training examples that are nearest to Xq
  - Return

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

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

PROGRAM:

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'Class']
dataset = pd.read_csv("Iris.csv", names=names, header=0)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(Xtrain, ytrain)
ypred = classifier.predict(Xtest)
print("\n..............................................................
...")
print('{:<25} {:<25} {}'.format('Original Label', 'Predicted Label',
'Correct/Wrong'))
```

```
print("\n..............................................................
...")
for i, label in enumerate(ytest):
    print('{:<25} {:<25} {}'.format(label, ypred[i], 'Correct' if label
== ypred[i] else 'Wrong'))
print("\n..............................................................
...")
print("\nConfusion Matrix:\n", metrics.confusion_matrix(ytest, ypred))
print("\n..............................................................
...")
print("\nClassification Report:\n",
metrics.classification_report(ytest, ypred))
print("\n..............................................................
...")
print('Accuracy of the classifier is
{:.2f}'.format(metrics.accuracy_score(ytest, ypred)))
print("\n..............................................................
...")
```

TRAINING DATA:
Note: Download Data from Kaggle

	A	B	C	D	E	F	G
1	Id	SepalLeng	SepalWidt	PetalLengt	PetalWidtl	Species	
2	1	5.1	3.5	1.4	0.2	Iris-setosa	
3	2	4.9	3	1.4	0.2	Iris-setosa	
4	3	4.7	3.2	1.3	0.2	Iris-setosa	
5	4	4.6	3.1	1.5	0.2	Iris-setosa	
6	5	5	3.6	1.4	0.2	Iris-setosa	
7	6	5.4	3.9	1.7	0.4	Iris-setosa	
8	7	4.6	3.4	1.4	0.3	Iris-setosa	
9	8	5	3.4	1.5	0.2	Iris-setosa	
10	9	4.4	2.9	1.4	0.2	Iris-setosa	

OUTPUT:

```
   sepal-length  sepal-width  petal-length  petal-width
1           5.1          3.5           1.4          0.2
2           4.9          3.0           1.4          0.2
3           4.7          3.2           1.3          0.2
4           4.6          3.1           1.5          0.2
5           5.0          3.6           1.4          0.2

.................................................................
Original Label              Predicted Label             Correct/Wrong

.................................................................
Iris-versicolor             Iris-versicolor             Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-setosa                 Iris-setosa                 Correct
Iris-setosa                 Iris-setosa                 Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-versicolor             Iris-virginica              Wrong
Iris-virginica              Iris-virginica              Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-virginica              Iris-virginica              Correct
Iris-setosa                 Iris-setosa                 Correct
Iris-setosa                 Iris-setosa                 Correct
Iris-versicolor             Iris-versicolor             Correct
Iris-virginica              Iris-virginica              Correct
Iris-virginica              Iris-virginica              Correct

.................................................................
```

```
.................................................................

Confusion Matrix:
 [[6 0 0]
 [0 7 0]
 [0 0 2]]

.................................................................

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         6
Iris-versicolor       1.00      1.00      1.00         7
 Iris-virginica       1.00      1.00      1.00         2

       accuracy                           1.00        15
      macro avg       1.00      1.00      1.00        15
   weighted avg       1.00      1.00      1.00        15


.................................................................
Accuracy of the classifier is 1.00

.................................................................
```

PROGRAM 9

Aim: Understand and analyse the concept of Regression algorithm techniques.
Program: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.

ALGORITHM:

1. Read the Given data Sample to X and the curve (linear or nonlinear) to Y.
2. Set the value for Smoothing parameter or Free parameter say t.
3. Set the bias /Point of interest set x0 which is a subset of X.
4. Determine the weight matrix using:

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

5. Determine the value of model term parameter ß using:

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

6. Prediction = x0*ß

TRAINING DATA:

	A	B	C	D	E	F	G	H
1	total_bill	tip	sex	smoker	day	time	size	
2	16.99	1.01	Female	No	Sun	Dinner	2	
3	10.34	1.66	Male	No	Mon	Dinner	3	
4	21.01	3.5	Male	No	Tue	Dinner	3	
5	23.68	3.31	Male	No	Wed	Dinner	2	
6	24.59	3.61	Female	No	Thu	Dinner	4	
7	25.29	4.71	Male	No	Fri	Dinner	4	
8	8.77	2	Male	No	Sat	Dinner	2	
9	26.88	3.12	Male	No	Sun	Dinner	4	
10	15.04	1.96	Male	No	Mon	Dinner	2	
11								

PROGRAM:

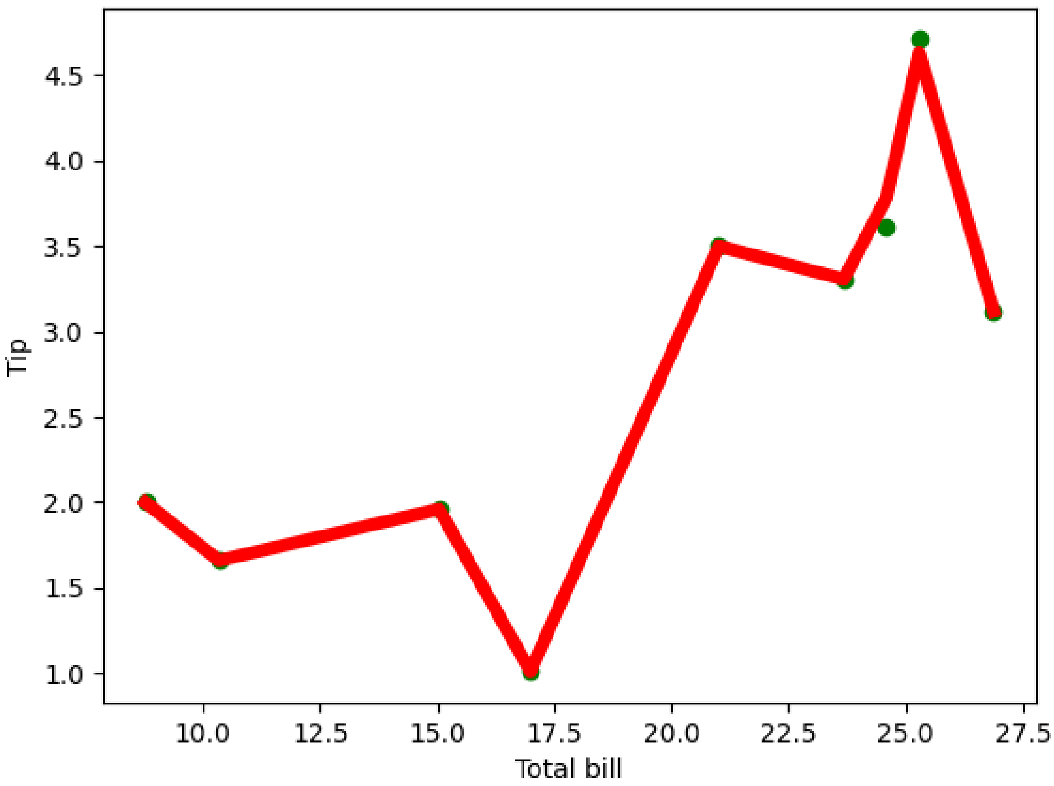
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

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(np.sum(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)).item()
    return ypred
data=pd.read_csv('9.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:



4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Tip
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
Total bill

PROGRAM 10

Aim: Implement and demonstrate classification algorithm using Support vector machine Algorithm.

Program: Implement and demonstrate the working of SVM algorithm for classification.

PROGRAM:

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score
iris = datasets.load_iris()
X = iris.data
y = iris.target
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)
y_pred = svm_classifier.predict(X_test)
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

OUTPUT:

```
Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45


Accuracy Score:
0.9777777777777777
```