**Program 1**

```
import csv
with open('play_tennis.csv','r')as f:
 reader=csv.reader(f)
 your_list=list(reader)
 k=0
 h=[['0','0','0','0','0','0']]
 for i in your_list:
     print("for the training sample",(k))
     if i[-1] =="True":
         j=0
         for x in i:
             if x !="True":
                 if x !=h[0][j] and h[0][j]=='0':
                     h[0][j]=x
                 elif x!=h[0][j] and h[0][j]!='0':
                     h[0][j]='?'
             j=j+1
     k=k+1
     print("the hypothesis is:",h)
 print("the maximally specific hypothesis is",h)
```

**Program 2**

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

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

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

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

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

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)
print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

## program 3

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

# Load data
data = pd.read_csv("play.csv")
features = [feat for feat in data if feat != "answer"]

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

# Calculate entropy
def entropy(examples):
    pos, neg = sum(examples["answer"] == "yes"), sum(examples["answer"] == "no")
    if pos == 0 or neg == 0:
        return 0.0
    p, n = pos / (pos + neg), neg / (pos + neg)
    return -(p * math.log(p, 2) + n * math.log(n, 2))

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

# ID3 algorithm
def ID3(examples, attrs):
    max_feat = max(attrs, key=lambda attr: info_gain(examples, attr))
    root = Node(value=max_feat)
    for u in np.unique(examples[max_feat]):
        subdata = examples[examples[max_feat] == u]
        child = Node(value=u)
        if entropy(subdata) == 0:
            child.isLeaf = True
            child.pred = subdata["answer"].iloc[0]
        else:
            child.children.append(ID3(subdata, [a for a in attrs if a != max_feat]))
        root.children.append(child)
    return root
# Print the decision tree
def printTree(root, depth=0):
    print("\t" * depth + root.value, "->" if root.isLeaf else "", root.pred if root.isLeaf else "")
    for child in root.children:
        printTree(child, depth + 1)

# Classify new data
def classify(root, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print("Predicted Label for new example:", child.pred)
            else:
                classify(child.children[0], new)
# Run the ID3 algorithm
root = ID3(data, features)
print("Decision Tree:")
printTree(root)
# Classification of a new example
new_example = {"outlook": "sunny", "temperature": "hot", "humidity": "normal", "wind": "strong"}
classify(root, new_example)
```

## PROGRAM 4

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) #maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5 #Setting training iterations
lr=0.1 #Setting learning rate

inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)
    #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad

    wout += hlayer_act.T.dot(d_output) *lr   # dotproduct of nextlayererror and currentlayerop
    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)
```

**Program 5 6**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

msg=pd.read_csv('6-Dataset.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

#splitting the dataset into train and test data
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 the words or Tokens in the text documents
cv = CountVectorizer()
```

```
xtrain_dtm = cv.fit_transform(xtrain)
xtest_dtm=cv.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(cv.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=cv.get_feature_names())

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

#printing accuracy, Confusion matrix, Precision and Recall
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))
```

## PROGRAM 7

```
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

heartDisease = pd.read_csv('7-dataset.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= BayesianModel([('age','heartdisease'),('gender','heartdisease'),
('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','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={'restecg':1})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

## ***PROGRAM 8***

```
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


names = [‘Sepal_Length’, ‘Sepal_Width’, ‘Petal_Length’, ‘Petal_Width’, ‘Class’]
dataset = pd.read_csv(“8-dataset.csv”, names=names)
X = dataset.iloc[:, :-1]
label = {‘Iris-setosa’:0, ’Iris-versicolor’: 1, ‘Iris-virginica’: 2}
y = [label[c] for c in dataset.iloc[:, -1]]
plt.figure(figsize=(14,7))
colormap=np.array([‘red’, ‘lime’, ‘black’])

# REAL PLOT
plt.subplot(1,3,1)
plt.title(&#39;Real&#39;)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])




# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title(‘KMeans’)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print(‘The accuracy score of K-Mean:’,metrics.accuracy_score(y, model.labels_))
print(‘The Confusion matrixof K-Mean:\n’,metrics.confusion_matrix(y, model.labels_))

# GMM PLOT
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.Petal_Length,X.Petal_Width,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))
```

## PROGRAM 10

```
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

# load data points
data = pd.read_csv('10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)


#preparing and add 1 in bill
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))

#set k here
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();
```