**PROGRAM 1:**

import csv

with open ('./enjoysport1.csv','r') as f:

    reader = csv.reader(f)

    data = list(reader)

for row in data:

    print (row)

aL=len(data[0])-1

h=aL\*['0']

print (h)

k=0

for row in data:

    if row[-1] == 'yes':

        j=0

        for col in row:

            if col != "yes":

                if col!=h[j] and h[j]=='0':

                    h[j]=col

                elif col!=h[j] and h[j]!='0':

                    h[j]='?'

            j=j+1

    print('h',k,h)

    k=k+1

print ('h',k-1,h)

**PROGRAM 2:**

import csv

with open('D://enjoysport1.csv', 'r') as f:

    reader = csv.reader(f)

    data = list(reader)

#Training data from CSV file

print("Training data")

for row in data:

    print(row)

print("--------------------------------------")

attr\_len=len(data[0])-1

#initialize Specific and General Hypothesis

S = ['0']\*attr\_len

G = ['?']\*attr\_len

temp=[] # altered G

print("The Hypothesis are")

print("S=",S)

print("G=",G)

print("--------------------------------------")

for row in data:

    if row[-1] == 'yes':

        j = 0

        for col in row:

            if col != 'yes':

                if col != S[j] and S[j] == '0':

                    S[j] = col

                elif col != S[j] and S[j] != '0':

                    S[j] = '?'

            j = j + 1

        for j in range(0,attr\_len):

            for k in temp:

                if k[j] != S[j] and k[j] != '?':

                    temp.remove(k)

    if row[-1]=='no':

        j = 0

        for col in row:

            if col != 'no':

                if col!= S[j] and S[j] != '?':

                    G[j]=S[j]

                    temp.append(G)

                    G=['?']\*attr\_len

            j =j + 1

    print("S=",S)

    if len(temp)==0:

        print("G=",G)

    else:

        print("G=",temp)

    print('-----------------------------------')

**PROGRAM 3:**

import pandas as pd

df = pd.read\_csv('D://id3.csv')

print("\n Input Data Set is:\n", df)

t = df.keys()[-1]

print('Target Attribute is: ', t)

# Get the attribute names from input dataset

attribute\_names = list(df.keys())

#Remove the target attribute from the attribute names list

attribute\_names.remove(t)

print('Predicting Attributes: ', attribute\_names)

#Function to calculate the entropy of collection S

import math

def entropy(probs):

    return sum( [-prob\*math.log(prob, 2) for prob in probs])

#Function to calulate the entropy of the given Data Sets/List with

#respect to target attributes

def entropy\_of\_list(ls,value):

    from collections import Counter

    cnt = Counter(x for x in ls)# Counter calculates the propotion of class

    print('Target attribute class count(Yes/No)=',dict(cnt))

    total\_instances = len(ls)

    print("Total no of instances/records associated with {0} is: {1}".format(value,total\_instances ))

    probs = [x / total\_instances for x in cnt.values()]  # x means no of YES/NO

    return entropy(probs) # Call Entropy

def information\_gain(df, split\_attribute, target\_attribute,battr):

    print("\n\n-----Information Gain Calculation of ",split\_attribute, " --------")

    df\_split = df.groupby(split\_attribute) # group the data based on attribute values

    glist=[]

    for gname,group in df\_split:

        print('Grouped Attribute Values \n',group)

        glist.append(gname)

    nobs = len(df.index)

    df\_agg1=df\_split.agg({target\_attribute:lambda x:entropy\_of\_list(x, glist.pop())})

    df\_agg2=df\_split.agg({target\_attribute :lambda x:len(x)/nobs})

    df\_agg1.columns=['Entropy']

    df\_agg2.columns=['Proportion']

    # Calculate Information Gain:

    new\_entropy = sum( df\_agg1['Entropy'] \* df\_agg2['Proportion'])

    if battr !='S':

        old\_entropy = entropy\_of\_list(df[target\_attribute],'S-'+df.iloc[0][df.columns.get\_loc(battr)])

    else:

        old\_entropy = entropy\_of\_list(df[target\_attribute],battr)

    return old\_entropy - new\_entropy

def id3(df, target\_attribute, attribute\_names, default\_class=None,default\_attr='S'):

    from collections import Counter

    cnt = Counter(x for x in df[target\_attribute])# class of YES /NO

    ## First check: Is this split of the dataset homogeneous?

    if len(cnt) == 1:

        return next(iter(cnt))  # next input data set, or raises StopIteration when EOF is hit.

    ## Second check: Is this split of the dataset empty? if yes, return a default value

    elif df.empty or (not attribute\_names):

        return default\_class  # Return None for Empty Data Set

    ## Otherwise: This dataset is ready to be divided up!

    else:

        # Get Default Value for next recursive call of this function:

        default\_class = max(cnt.keys()) #No of YES and NO Class

        # Compute the Information Gain of the attributes:

        gainz=[]

        for attr in attribute\_names:

            ig= information\_gain(df, attr, target\_attribute,default\_attr)

            gainz.append(ig)

            print('Information gain of ',attr,' is : ',ig)

        index\_of\_max = gainz.index(max(gainz))               # Index of Best Attribute

        best\_attr = attribute\_names[index\_of\_max]            # Choose Best Attribute to split on

        print("\nAttribute with the maximum gain is: ", best\_attr)

        # Create an empty tree, to be populated in a moment

        tree = {best\_attr:{}} # Initiate the tree with best attribute as a node

        remaining\_attribute\_names =[i for i in attribute\_names if i != best\_attr]

        # Split dataset-On each split, recursively call this algorithm.Populate the empty tree with subtrees, which

        # are the result of the recursive call

        for attr\_val, data\_subset in df.groupby(best\_attr):

            subtree = id3(data\_subset,target\_attribute, remaining\_attribute\_names,default\_class,best\_attr)

            tree[best\_attr][attr\_val] = subtree

        return tree

from pprint import pprint

tree = id3(df,t,attribute\_names)

print("\nThe Resultant Decision Tree is:")

pprint(tree)

**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)

    #how much hidden layer wts 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

    bout += np.sum(d\_output, axis=0,keepdims=True) \*lr

    wh += X.T.dot(d\_hiddenlayer) \*lr

    bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr

print("Input: \n" ,X)

print("Actual Output: \n" ,y)

print("Predicted Output: \n" ,output)

**PROGRAM 5:**

import numpy as np

import math

import csv

def read\_data(filename):

    with open(filename,"r") as csvfile:

        datareader=csv.reader(csvfile)

        traindata=list(datareader)

    metadata=traindata[0]    #attributes name

    traindata=traindata[1:]  #training examples

    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 classify(data,test):

    totalsize=data.shape[0]

    print("\n")

    print("training data size=",totalsize)

    print("test data size=",test.shape[0])

    countyes=0

    countno=0

    probyes=0

    probno=0

    print("\n")

    print("target \t count \t probolaity")

    for x in range(data.shape[0]):

        if data[x,-1] =='yes':

            countyes+=1

        if data[x,-1] =='no':

            countno+=1

    probYes=countyes/totalsize

    probNo=countno/totalsize

    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 \t prediction \t 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]):

                #how many times appeared with no

                if test[t,k]==data[j,k] and data[j,-1]=='no':

                    count0+=1

                #how many times appeared with yes

                if test[t,k]==data[j,k] and data[j,-1]=='yes':

                    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\t",predict," \t\t",test[t,-1])

        if predict==test[t,-1]:

            accuracy+=1

    finalaccuracy=(accuracy/test.shape[0])\*100

    print("Accuracy=",finalaccuracy,"%")

metadata,traindata=read\_data("D://enjoysport.csv")

print("\n The attribute names of training data are:",metadata)

splitratio=0.7

trainset, testset=splitdataset(traindata, splitratio)

training=np.array(trainset)

print("\n The Training data set are:")

for x in training:

    print(x)

testing=np.array(testset)

print("\n The Test data set are:")

for x in testing:

    print(x)

classify(training, testing)

**PROGRAM 6:**

import pandas as pd

msg=pd.read\_csv('D:\\data6.csv',names=['message','label']) *#Tabular form data*

print('Total instances in the dataset:',msg.shape[0])

msg['labelnum']=msg.label.map({'pos':1,'neg':0})

print("Dataset:")

print(msg)

X=msg.message

Y=msg.labelnum

*# 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('\nDataset is split into Training and Testing samples')

print('\nTotal training instances :', ytrain.shape[0])

*#print(xtrain)*

print('\nTotal testing instances :', ytest.shape[0])

*#print(xtest)*

*# Output of count vectoriser is a sparse matrix*

*# CountVectorizer - stands for 'feature extraction'*

from sklearn.feature\_extraction.text import CountVectorizer

count\_vect = CountVectorizer()

xtrain\_dtm = count\_vect.fit\_transform(xtrain) *#Sparse matrix*

xtest\_dtm = count\_vect.transform(xtest)

print('\nTotal features extracted using CountVectorizer:',xtrain\_dtm.shape[1])

*#print(xtrain\_dtm.toarray())*

print('\nThe words or Tokens in the text documents\n')

print(count\_vect.get\_feature\_names())

*# 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)

print("\nPredicted")

print('==================')

print(predicted)

print("\nActual")

print('==================')

print(list(ytest))

*#printing accuracy metrics*

from sklearn import metrics

print('\nAccuracy metrics')

print('==================')

print('Accuracy of the classifer is',metrics.accuracy\_score(ytest,predicted))

print('Recall :',metrics.recall\_score(ytest,predicted), '\nPrecison :',metrics.precision\_score(ytest,predicted))

print('Confusion matrix')

print('==================')

print(metrics.confusion\_matrix(ytest,predicted))

**PROGRAM 8:**

import matplotlib.pyplot as plt

from sklearn import datasets

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']

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

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

# Plot the Original Classifications

plt.subplot(1, 3, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40)

plt.title('Real Classification')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

# Plot the Models Classifications

#KMeans

from sklearn.cluster import KMeans

model = KMeans(n\_clusters=3)

model.fit(X)

plt.subplot(1, 3, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40)

plt.title('K Mean Classification')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

print('The accuracy score of K-Mean: ',sm.accuracy\_score(y, model.labels\_))

print('The Confusion matrixof K-Mean: ',sm.confusion\_matrix(y, model.labels\_))

#EM(GMM)

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=3)

gmm.fit(X)

y\_gmm = gmm.predict(X)

plt.subplot(1, 3, 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))

**PROGRAM 9:**

from sklearn import datasets

iris=datasets.load\_iris()

iris\_data=iris.data

iris\_labels=iris.target

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(iris\_data,iris\_labels,test\_size=0.30)

from sklearn.neighbors import KNeighborsClassifier

classifier=KNeighborsClassifier(n\_neighbors=5)

classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test)

#print(iris.target\_names)

target\_names = iris.target\_names

for pred,actual in zip(y\_pred,y\_test):

     print("Prediction is "+str(target\_names[pred])+", Actual is "+str(target\_names[actual]))

print("\*"\*50,"\nClassification Score:", classifier.score(x\_test,y\_test),"\n","\*"\*50)

from sklearn.metrics import classification\_report,confusion\_matrix

print('Confusion matrix is as follows')

print(confusion\_matrix(y\_test,y\_pred))

print('Accuracy Metrics')

print(classification\_report(y\_test,y\_pred))