**Program 1**

import pandas as pd  
import numpy as np  
data = pd.read\_csv('data.csv')  
concept = np.array(data)[:,:-1]  
target = np.array(data)[:,-1]  
def train(con,tar):  
    for i , val in enumerate(tar):  
        if val == 'yes':  
            specific\_h = con[i].copy()  
            break  
    for i, val in enumerate(con):  
        if tar[i] == 'yes':  
            for x in range(len(specific\_h)):  
                if val[x]!=specific\_h[x]:  
                    specific\_h[x] = '?'  
                else:  
                    pass  
    return specific\_h  
print(train(concept,target))

OUTPUT

output:['sunny' 'warm' 'high' 'strong' '?' '?']

DATASET(data.csv)

a,b,c,d,e,f,g  
sunny,warm,normal,strong,warm,same,yes  
sunny,warm,high,strong,warm,same,yes  
rainy,cold,high,strong,warm,change,no  
sunny,warm,high,strong,cool,change,yes

**program 2**

import numpy as np  
import pandas as pd  
data = pd.read\_csv('data.csv')  
concepts = np.array(data.iloc[:,0:-1])  
target = np.array(data.iloc[:,-1])  
def learn(concepts,target):  
    count = 0  
    first = ['?','?','?','?','?','?']  
    for i, val in enumerate(target):  
        if val == 'yes':  
            #print(specific\_h)  
            break  
    specific\_h = concepts[i].copy()  
    generic\_h = [["?" for i in range(len(specific\_h))] for i in range (len(specific\_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] = "?"  
                  generic\_h[x][x] = "?"      
        if target[i] == "no":  
            for x in range(len(specific\_h)):  
                if h[x] != specific\_h[x]:  
                    generic\_h[x][x] = specific\_h[x]  
                else:  
                    generic\_h[x][x] == "?"  
        if generic\_h[x][x] == "?":  
            print(f'S{count} : {specific\_h}')  
            print(f'G{count} : {first}')  
            count+=1      
        else:  
            print(f'S{count} : {specific\_h}')  
            print(f'G{count} : {generic\_h}')  
            for x in range(len(generic\_h)):  
                first[x] = generic\_h[x]  
            count += 1        
    indices=[i for i, val in enumerate(generic\_h)if val == ['?','?','?','?','?','?']]  
    for i in indices:  
        generic\_h.remove(['?','?','?','?','?','?'])  
    return specific\_h, generic\_h  
  
s\_final,g\_final = learn(concepts,target)  
print("final s:",s\_final,sep="\n")  
print("final g:",g\_final,sep="\n")

output

S0 : ['sunny' 'warm' 'high' 'strong' 'warm' 'same']

G0 : ['?', '?', '?', '?', '?', '?']

S1 : ['sunny' 'warm' 'high' 'strong' 'warm' 'same']

G1 : [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

S2 : ['sunny' 'warm' 'high' 'strong' '?' '?']

G2 : [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

final s:

['sunny' 'warm' 'high' 'strong' '?' '?']

final g:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Same dataset 2(as of pgm1)

Program 3

import csv  
def read\_data(filename):  
    with open(filename, 'r') as csvfile:  
        datareader = csv.reader(csvfile, delimiter=',')  
        headers = next(datareader)  
        metadata = []  
        traindata = []  
        for name in headers:  
            metadata.append(name)  
        for row in datareader:  
            traindata.append(row)  
    return (metadata, traindata)  
  
import numpy as np  
import math  
#from data\_loader import read\_data  
class Node:  
    def \_\_init\_\_(self, attribute):  
        self.attribute = attribute  
        self.children = []  
        self.answer = ""  
  
    def \_\_str\_\_(self):  
        return self.attribute  
  
def subtables(data, col, delete):  
    dict = {}  
    items = np.unique(data[:, col])  
    count = np.zeros((items.shape[0], 1), dtype=np.int32)      
    for x in range(items.shape[0]):  
        for y in range(data.shape[0]):  
            if data[y, col] == items[x]:  
                count[x] += 1  
    for x in range(items.shape[0]):  
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")  
        pos = 0  
        for y in range(data.shape[0]):  
            if data[y, col] == items[x]:  
                dict[items[x]][pos] = data[y]  
                pos += 1      
        if delete:  
            dict[items[x]] = np.delete(dict[items[x]], col, 1)  
    return items, dict      
def entropy(S):  
    items = np.unique(S)  
  
  
    if items.size == 1:  
        return 0  
    counts = np.zeros((items.shape[0], 1))  
    sums = 0  
    for x in range(items.shape[0]):  
        counts[x] = sum(S == items[x]) / (S.size \* 1.0)  
    for count in counts:  
        sums += -1 \* count \* math.log(count, 2)  
    return sums  
  
def gain\_ratio(data, col):  
    items, dict = subtables(data, col, delete=False)  
    total\_size = data.shape[0]  
    entropies = np.zeros((items.shape[0], 1))  
    intrinsic = np.zeros((items.shape[0], 1))  
    for x in range(items.shape[0]):  
        ratio = dict[items[x]].shape[0]/(total\_size \* 1.0)  
        entropies[x] = ratio \* entropy(dict[items[x]][:, -1])  
        intrinsic[x] = ratio \* math.log(ratio, 2)  
  
    total\_entropy = entropy(data[:, -1])  
    iv = -1 \* sum(intrinsic)  
    for x in range(entropies.shape[0]):  
        total\_entropy -= entropies[x]  
    return total\_entropy / iv  
  
def create\_node(data, metadata):  
    if (np.unique(data[:, -1])).shape[0] == 1:  
        node = Node("")  
        node.answer = np.unique(data[:, -1])[0]  
        return node  
    gains = np.zeros((data.shape[1] - 1, 1))  
    for col in range(data.shape[1] - 1):  
        gains[col] = gain\_ratio(data, col)  
    split = np.argmax(gains)  
  
  
    node = Node(metadata[split])      
    metadata = np.delete(metadata, split, 0)  
    items, dict = subtables(data, split, delete=True)  
    for x in range(items.shape[0]):  
        child = create\_node(dict[items[x]], metadata)  
        node.children.append((items[x], child))  
    return node          
  
def empty(size):  
    s = ""  
    for x in range(size):  
        s += "   "  
    return s  
  
def print\_tree(node, level):  
    if node.answer != "":  
        print(empty(level), node.answer)  
        return  
    print(empty(level), node.attribute)  
    for value, n in node.children:  
        print(empty(level + 1), value)  
        print\_tree(n, level + 2)  
metadata, traindata = read\_data("tennis.data")  
data = np.array(traindata)  
node = create\_node(data, metadata)  
print\_tree(node, 0)

**dataset**

tennis.data

outlook,temperature,humidity,wind,answer

sunny,hot,high,weak,no

sunny,hot,high,strong,no

overcast,hot,high,weak,yes

rain,mild,high,weak,yes

rain,cool,normal,weak,yes

rain,cool,normal,strong,no

overcast,cool,normal,strong,yes

sunny,mild,high,weak,no

sunny,cool,normal,weak,yes

rain,mild,normal,weak,yes

sunny,mild,normal,strong,yes

overcast,mild,high,strong,yes

overcast,hot,normal,weak,yes

rain,mild,high,strong,no

**output**

outlook

overcast

b'yes'

rain

wind

b'strong'

b'no'

b'weak'

b'yes'

sunny

humidity

b'high'

b'no'

b'normal'

b'yes'

**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)  
y=y/100  
def sigmoid(x):  
    return 1/(1+np.exp(-x))  
def sigmoid\_grad(x):  
    return x\*(1-x)  
epoch=1000  
eta=0.2  
input\_neurons=2  
hidden\_neurons=2  
output\_neurons=2  
wh=np.random.uniform(size=(input\_neurons,hidden\_neurons))  
bh=np.random.uniform(size=(1,hidden\_neurons))  
wout=np.random.uniform(size=(hidden\_neurons,output\_neurons))  
bout=np.random.uniform(size=(1,output\_neurons))  
for i in range(epoch):  
    h\_ip=np.dot(X,wh)+bh  
    h\_act=sigmoid(h\_ip)  
    o\_ip=np.dot(h\_act,wout)+bout  
    output=sigmoid(o\_ip)  
    Eo=y-output  
    outgrad=sigmoid\_grad(output)  
    d\_output=Eo\*outgrad  
    Eh=d\_output.dot(wout.T)  
    hiddengrad=sigmoid\_grad(h\_act)  
    d\_hidden=Eh\*hiddengrad  
    wout +=h\_act.T.dot(d\_output)\*eta  
    wh+=X.T.dot(d\_hidden)\*eta  
print("Normalized Input:\n"+str(X))  
print("Actual Output:\n"+str(y))  
print("Predicted Output:\n",output)

**output(no dataset )**

Normalized Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.89432494 0.89425803]

[0.88011354 0.87994543]

[0.89534233 0.89559409]]

**program 5**

import csv  
import random  
import math  
   
def loadCsv(filename):  
    lines = csv.reader(open(filename))  
    dataset = list(lines)  
    for i in range(len(dataset)):  
        dataset[i] = [float(x) for x in dataset[i]]  
    return dataset  
def splitDataset(dataset, splitRatio):  
    trainSize = int(len(dataset) \* splitRatio)  
    trainSet = []  
    copy = list(dataset)  
    while len(trainSet) < trainSize:  
        index = random.randrange(len(copy))  
        trainSet.append(copy.pop(index))  
    return [trainSet, copy]  
def separateByClass(dataset):  
    separated = {}  
    for i in range(len(dataset)):  
        vector = dataset[i]  
        if (vector[-1] not in separated):  
            separated[vector[-1]] = []  
        separated[vector[-1]].append(vector)  
    return separated  
def mean(numbers):  
    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):  
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)]  
    del summaries[-1]  
    return summaries  
def summarizeByClass(dataset):  
    separated = separateByClass(dataset)  
    summaries = {}  
    for classValue, instances in separated.items():  
        summaries[classValue] = summarize(instances)  
    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 = {}  
    for classValue, classSummaries in summaries.items():  
        probabilities[classValue] = 1  
        for i in range(len(classSummaries)):  
            mean, stdev = classSummaries[i]  
            x = inputVector[i]  
            probabilities[classValue] \*= calculateProbability(x, mean, stdev)  
    return probabilities  
def predict(summaries, inputVector):  
    probabilities = calculateClassProbabilities(summaries, inputVector)  
    bestLabel, bestProb = None, -1  
    for classValue, probability in probabilities.items():  
        if bestLabel is None or probability > bestProb:  
            bestProb = probability  
            bestLabel = classValue  
    return bestLabel   
def getPredictions(summaries, testSet):  
    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 = 'data5.csv'  
    splitRatio = 0.67  
    dataset = loadCsv(filename)  
    trainingSet, testSet = splitDataset(dataset, splitRatio)  
    print(f"Split {len(dataset)} rows into train={len(trainingSet)} and test={len(testSet)} rows")  
    summaries = summarizeByClass(trainingSet)  
    predictions = getPredictions(summaries, testSet)  
    accuracy = getAccuracy(testSet, predictions)  
    print(f'Accuracy: {accuracy}%')  
main()

output(give diabetes dataset or any other dataset)

Split 768 rows into train=514 and test=254 rows

Accuracy: 74.80314960629921%

PROGRAM 6

import pandas as pd  
msg=pd.read\_csv('naivetext1.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)  
from sklearn.model\_selection import train\_test\_split  
xtrain,xtest,ytrain,ytest=train\_test\_split(X,y)  
print(xtest.shape)  
print(xtrain.shape)  
print(ytest.shape)  
print(ytrain.shape)  
from sklearn.feature\_extraction.text import CountVectorizer  
count\_vect=CountVectorizer()  
xtrain\_dtm=count\_vect.fit\_transform(xtrain)  
xtest\_dtm=count\_vect.transform(xtest)  
from sklearn.naive\_bayes import MultinomialNB  
clf=MultinomialNB().fit(xtrain\_dtm,ytrain)  
predicted=clf.predict(xtest\_dtm)  
from sklearn import metrics  
print('Accuracy metrics')  
print('Accuracy of the classifer is',metrics.accuracy\_score(ytest,predicted))  
print('Confusion matrix')  
print(metrics.confusion\_matrix(ytest,predicted))  
print('Recall and Precison')  
print(metrics.recall\_score(ytest,predicted))  
print(metrics.precision\_score(ytest,predicted))

**dataset.....**

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

Output

The dimensions of the dataset (18, 2)

0 I love this sandwich

1 This is an amazing place

2 I feel very good about these beers

3 This is my best work

4 what an awesome view

5 I do not like this restaurant

6 I am tired of this stuff

7 I can't deal with this

8 He is my sworn enemy

9 My boss is horrible

10 This is an awesome place

11 I do not like the taste of this juice

12 I love to dance

13 I am sick and tired of this place

14 What a great holiday

15 That is a bad locality to stay

16 We will have good fun tomorrow

17 I went to my enemy's house today

Name: message, dtype: object

0 1

1 1

2 1

3 1

4 1

5 0

6 0

7 0

8 0

9 0

10 1

11 0

12 1

13 0

14 1

15 0

16 1

17 0

Name: labelnum, dtype: int64

(5,)

(13,)

(5,)

(13,)

Accuracy metrics

Accuracy of the classifer is 0.8

Confusion matrix

[[3 1]

[0 1]]

Recall and Precison

1.0

0.5

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

lines = list(csv.reader(open(r'C:\Users\MCA\Desktop\New folder\data7\_names.csv', 'r')))

attributes = lines[0]

heartDisease = pd.read\_csv(r'C:\Users\MCA\Desktop\New folder\data7\_heart.csv', names = attributes)

heartDisease = heartDisease.replace('?', np.nan)

model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])

print('\nLearning CPDs using Maximum Likelihood Estimators...');

model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

print('\nInferencing with Bayesian Network:')

HeartDisease\_infer = VariableElimination(model)

print('\n1.Probability of HeartDisease given Age=28')

q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={'age': 28})

print(q['heartdisease'])

print('\n2. Probability of HeartDisease given chol (Cholestoral) =100')

q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={'chol': 100})

print(q['heartdisease'])

dataset

data7\_names.csv ,data7\_heart.csv

output

Learning CPDs using Maximum Likelihood Estimators...

Inferencing with Bayesian Network:

1.Probability of HeartDisease given Age=28

+----------------+---------------------+

| heartdisease | phi(heartdisease) |

+================+=====================+

| heartdisease\_0 | 0.5932 |

+----------------+---------------------+

| heartdisease\_1 | 0.1714 |

+----------------+---------------------+

| heartdisease\_2 | 0.0811 |

+----------------+---------------------+

| heartdisease\_3 | 0.1138 |

+----------------+---------------------+

| heartdisease\_4 | 0.0405 |

+----------------+---------------------+

2. Probability of HeartDisease given chol (Cholestoral) =100

+----------------+---------------------+

| heartdisease | phi(heartdisease) |

+================+=====================+

| heartdisease\_0 | 1.0000 |

+----------------+---------------------+

| heartdisease\_1 | 0.0000 |

+----------------+---------------------+

| heartdisease\_2 | 0.0000 |

+----------------+---------------------+

| heartdisease\_3 | 0.0000 |

+----------------+---------------------+

| heartdisease\_4 | 0.0000 |

+----------------+---------------------+

PROGRAM 8

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

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=(14,14))

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

plt.subplot(2,2,1)

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

plt.title('Real Clusters')

plt.xlabel('Petal Length')

plt.ylabel('Petal width')

plt.subplot(2,2,2)

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

plt.title('K-Means Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.subplot(2,2,2)

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

plt.title('K-Means Clustering')

plt.ylabel('Petal Width')

from sklearn import preprocessing

scaler=preprocessing.StandardScaler()

scaler.fit(X)

xsa=scaler.transform(X)

xs=pd.DataFrame(xsa,columns=X.columns)

from sklearn.mixture import GaussianMixture

gmm=GaussianMixture(n\_components=3)

gmm.fit(xs)

gmm\_y=gmm.predict(xs)

plt.subplot(2,2,3)

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

plt.title('GMM Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal width')

print('Observation:The GMM using EM algo based clustering matched the true labels more closely than KMeans.')

output

3 graphs

PROGRAM 9

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

import pandas as pd

import numpy as np

iris=datasets.load\_iris()

iris\_data=iris.data

iris\_label=iris.target

x\_train,x\_test,y\_train,y\_test=train\_test\_split(iris\_data,iris\_label)

classifier=KNeighborsClassifier(n\_neighbors=5)

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

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

print('Confusion matrix is as follows')

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

print('Accuracy Metrics')

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

output(no dataset)

Confusion matrix is as follows

[[13 0 0]

[ 0 11 1]

[ 0 0 13]]

Accuracy Metrics

precision recall f1-score support

0 1.00 1.00 1.00 13

1 1.00 0.92 0.96 12

2 0.93 1.00 0.96 13

avg / total 0.98 0.97 0.97 38

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

def graphPlot(X,ypred):

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();

data = pd.read\_csv('data10.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,3)

graphPlot(X,ypred)

dataset

data10\_tips.csv

output

one single graph