1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

```
import csv
a = []

with open('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("")
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)
```

<u>Output</u>:

```
[['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', '?', '?']

2. 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.

```
import numpy as np
import pandas as pd

data = pd.read_csv('enjoysport1.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")
```

<u>Output</u>:

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]




For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]




For Loop Starts
If instance is Negative
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
```

```
For Loop Starts
If instance is Positive
  steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]




Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

3. 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.

```
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

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("id3.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("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
    rain
       Wind
          strong
             no
          weak
             yes
    sunny
       Humidity
```

```
            normal
               yes
            high
               no
      overcast
         yes
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
```

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

```
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)
```

<u>Output</u>:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.83156396]
 [0.82074463]
 [0.83540644]]
```

5. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd
from pandas import DataFrame

df_golf = pd.read_csv(r"golf.csv")
print(df_golf)

attribute_names = list(df_golf.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('label')
print("Predicting Attributes:", attribute_names)
table=dict()
priorProb=dict()


train=df_golf.sample(frac=0.8,random_state=100) #random state is a seed value
test=df_golf.drop(train.index)

print("ddddddddddddddddddddddddddddddd")
print(train)
print("dddddddddddddddddddddddddddddddd")
print(test)
print("dddddddddddddddddddddddd")


for attr_val, data_subset in train.groupby("label"):
    from collections import Counter
    valueCount = dict()
    count=0
    for attr_value in attribute_names:
        cnt = Counter(x for x in data_subset[attr_value])
        count=sum(cnt.values())
        valueCount[attr_value]=dict(cnt)
        print("value count", valueCount.values())
        print("counter:-",cnt)

    table[attr_val]=valueCount
    priorProb[attr_val]=count
print("********************************************")
from pprint import pprint

print("\n\nThe Resultant table is :\n")
pprint(table)
```

```
pprint(priorProb)

totalSize=test['label'].count()
correctPridictions=0
for k, row in test.iterrows():

    rowTuple=dict(row)
    print("print row tuple")
    pprint(rowTuple)
    postioriList=list()
    labelList=list()
    for label in table.keys():
        posteriori = 1.0
        print("RowTuple",rowTuple.keys())
        print("RowValues",rowTuple.values())
        for key in [x for x in rowTuple.keys() if x != 'label']:
            print(key, "label:",label)
            attributeValue=rowTuple.get(key)
            if attributeValue in table[label][key].keys():
                countList=table[label][key].values()
                #print("CountList:", countList)
                attributeCount=table[label][key][attributeValue]
                #print("CountList:",countList)
                #print("SumCountList",sum(countList))
                #print("AttributeCount:",attributeCount)
                #print("key:valuepair",key,":",rowTuple[key])
                posteriori=1.0*attributeCount/sum(countList)*posteriori

        posteriori=posteriori*priorProb[label]
        labelList.append(label)
        postioriList.append(posteriori)
        print(labelList)
        print(postioriList)
    maxProbInd = postioriList.index(max(postioriList))
    print(rowTuple['label'], "::::", labelList[maxProbInd])
    if rowTuple['label'] == labelList[maxProbInd]:
        print(rowTuple['label'],"::::",labelList[maxProbInd])
        correctPridictions=correctPridictions+1
        print("POSTERIORI OF:",label,"is:",posteriori)
print("Number of Correct Predictions : Number of Samples",correctPridictions,":",totalSize)
print("Accuracy:",100.0*correctPridictions/totalSize)
```

Output:

```
      id     outlook    temp humidity      wind label
0      1       Sunny'     Hot'      High'      Weak'     No'
1      2       Sunny'     Hot'      High'    Strong'     No'
2      3    Overcast'     Hot'      High'      Weak'    Yes'
3      4        Rain'    Mild'      High'      Weak'    Yes'
4      5        Rain'    Cool'    Normal'      Weak'    Yes'
5      6        Rain'    Cool'    Normal'    Strong'     No'
6      7    Overcast'    Cool'    Normal'    Strong'    Yes'
7      8       Sunny'    Mild'      High'      Weak'     No'
8      9       Sunny'    Cool'    Normal'      Weak'    Yes'
9     10        Rain'    Mild'    Normal'      Weak'    Yes'
10    11       Sunny'    Mild'    Normal'    Strong'    Yes'
11    12    Overcast'    Mild'      High'    Strong'    Yes'
12    13    Overcast'     Hot'    Normal'      Weak'    Yes'
13    14        Rain'    Mild'      High'    Strong'     No'
List of Attributes: ['id', 'outlook', 'temp', 'humidity', 'wind', 'label']
Predicting Attributes: ['id', 'outlook', 'temp', 'humidity', 'wind']
dddddddddddddddddddddddddddddd
      id     outlook    temp humidity      wind label
11    12    Overcast'    Mild'      High'    Strong'    Yes'
12    13    Overcast'     Hot'    Normal'      Weak'    Yes'
5      6        Rain'    Cool'    Normal'    Strong'     No'
1      2       Sunny'     Hot'      High'    Strong'     No'
9     10        Rain'    Mild'    Normal'      Weak'    Yes'
4      5        Rain'    Cool'    Normal'      Weak'    Yes'
6      7    Overcast'    Cool'    Normal'    Strong'    Yes'
2      3    Overcast'     Hot'      High'      Weak'    Yes'
0      1       Sunny'     Hot'      High'      Weak'     No'
10    11       Sunny'    Mild'    Normal'    Strong'    Yes'
7      8       Sunny'    Mild'      High'      Weak'     No'
ddddddddddddddddddddddddddddddd
      id outlook    temp humidity      wind label
3      4   Rain'    Mild'      High'      Weak'    Yes'
8      9  Sunny'    Cool'    Normal'      Weak'    Yes'
13    14   Rain'    Mild'      High'    Strong'     No'
dddddddddddddddddddddddd
value count dict_values([{6: 1, 2: 1, 1: 1, 8: 1}])
counter:- Counter({6: 1, 2: 1, 1: 1, 8: 1})
value count dict_values([{6: 1, 2: 1, 1: 1, 8: 1}, {"Rain'": 1, "Sunny'": 3}])
counter:- Counter({"Sunny'": 3, "Rain'": 1})
value count dict_values([{6: 1, 2: 1, 1: 1, 8: 1}, {"Rain'": 1, "Sunny'": 3}, {"Cool'": 1, "Hot'": 2, "Mild'": 1}])
counter:- Counter({"Hot'": 2, "Cool'": 1, "Mild'": 1})
```

```
value count dict_values([{6: 1, 2: 1, 1: 1, 8: 1}, {"Rain"': 1, "Sunny"': 3}, {"Cool"': 1, "Hot"':
2, "Mild"': 1}, {"Normal"': 1, "High"': 3}])
counter:- Counter({"High"': 3, "Normal"': 1})
value count dict_values([{6: 1, 2: 1, 1: 1, 8: 1}, {"Rain"': 1, "Sunny"': 3}, {"Cool"': 1, "Hot"':
2, "Mild"': 1}, {"Normal"': 1, "High"': 3}, {"Strong"': 2, "Weak"': 2}])
counter:- Counter({"Strong"': 2, "Weak"': 2})
value count dict_values([{12: 1, 13: 1, 10: 1, 5: 1, 7: 1, 3: 1, 11: 1}])
counter:- Counter({12: 1, 13: 1, 10: 1, 5: 1, 7: 1, 3: 1, 11: 1})
value count dict_values([{12: 1, 13: 1, 10: 1, 5: 1, 7: 1, 3: 1, 11: 1}, {"Overcast"': 4, "Rain"':
2, "Sunny"': 1}])
counter:- Counter({"Overcast"': 4, "Rain"': 2, "Sunny"': 1})
value count dict_values([{12: 1, 13: 1, 10: 1, 5: 1, 7: 1, 3: 1, 11: 1}, {"Overcast"': 4, "Rain"':
2, "Sunny"': 1}, {"Mild"': 3, "Hot"': 2, "Cool"': 2}])
counter:- Counter({"Mild"': 3, "Hot"': 2, "Cool"': 2})
value count dict_values([{12: 1, 13: 1, 10: 1, 5: 1, 7: 1, 3: 1, 11: 1}, {"Overcast"': 4, "Rain"':
2, "Sunny"': 1}, {"Mild"': 3, "Hot"': 2, "Cool"': 2}, {"High"': 2, "Normal"': 5}])
counter:- Counter({"Normal"': 5, "High"': 2})
value count dict_values([{12: 1, 13: 1, 10: 1, 5: 1, 7: 1, 3: 1, 11: 1}, {"Overcast"': 4, "Rain"':
2, "Sunny"': 1}, {"Mild"': 3, "Hot"': 2, "Cool"': 2}, {"High"': 2, "Normal"': 5}, {"Strong"': 3,
"Weak"': 4}])
counter:- Counter({"Weak"': 4, "Strong"': 3})
******************************************


The Resultant table is :

{"No"': {'humidity': {"High"': 3, "Normal"': 1},
                'id': {1: 1, 2: 1, 6: 1, 8: 1},
                'outlook': {"Rain"': 1, "Sunny"': 3},
                'temp': {"Cool"': 1, "Hot"': 2, "Mild"': 1},
                'wind': {"Strong"': 2, "Weak"': 2}},
  "Yes"': {'humidity': {"High"': 2, "Normal"': 5},
                 'id': {3: 1, 5: 1, 7: 1, 10: 1, 11: 1, 12: 1, 13: 1},
                 'outlook': {"Overcast"': 4, "Rain"': 2, "Sunny"': 1},
                 'temp': {"Cool"': 2, "Hot"': 2, "Mild"': 3},
                 'wind': {"Strong"': 3, "Weak"': 4}}}
{"No"': 4, "Yes"': 7}
print row tuple
{'humidity': "High"',
  'id': 4,
  'label': "Yes"',
  'outlook': "Rain"',
  'temp': "Mild"',
  'wind': "Weak"'}
RowTuple dict_keys(['id', 'outlook', 'temp', 'humidity', 'wind', 'label'])
RowValues dict_values([4, "Rain"', "Mild"', "High"', "Weak"', "Yes"'])
id label: No'
```

```
outlook label: No'
temp label: No'
humidity label: No'
wind label: No'
["No'"]
[0.09375]
RowTuple dict_keys(['id', 'outlook', 'temp', 'humidity', 'wind', 'label'])
RowValues dict_values([4, "Rain'", "Mild'", "High'", "Weak'", "Yes'"])
id label: Yes'
outlook label: Yes'
temp label: Yes'
humidity label: Yes'
wind label: Yes'
["No'", "Yes'"]
[0.09375, 0.13994169096209907]
Yes' :::: Yes'
Yes' :::: Yes'
POSTERIORI OF: Yes' is: 0.13994169096209907
print row tuple
{'humidity': "Normal'",
 'id': 9,
 'label': "Yes'",
 'outlook': "Sunny'",
 'temp': "Cool'",
 'wind': "Weak'"}
RowTuple dict_keys(['id', 'outlook', 'temp', 'humidity', 'wind', 'label'])
RowValues dict_values([9, "Sunny'", "Cool'", "Normal'", "Weak'", "Yes'"])
id label: No'
outlook label: No'
temp label: No'
humidity label: No'
wind label: No'
["No'"]
[0.09375]
RowTuple dict_keys(['id', 'outlook', 'temp', 'humidity', 'wind', 'label'])
RowValues dict_values([9, "Sunny'", "Cool'", "Normal'", "Weak'", "Yes'"])
id label: Yes'
outlook label: Yes'
temp label: Yes'
humidity label: Yes'
wind label: Yes'
["No'", "Yes'"]
[0.09375, 0.11661807580174925]
Yes' :::: Yes'
Yes' :::: Yes'
POSTERIORI OF: Yes' is: 0.11661807580174925
print row tuple
```

```
{'humidity': "High'",
  'id': 14,
  'label': "No'",
  'outlook': "Rain'",
  'temp': "Mild'",
  'wind': "Strong'"}
RowTuple dict_keys(['id', 'outlook', 'temp', 'humidity', 'wind', 'label'])
RowValues dict_values([14, "Rain'", "Mild'", "High'", "Strong'", "No'"])
id label: No'
outlook label: No'
temp label: No'
humidity label: No'
wind label: No'
["No'"]
[0.09375]
RowTuple dict_keys(['id', 'outlook', 'temp', 'humidity', 'wind', 'label'])
RowValues dict_values([14, "Rain'", "Mild'", "High'", "Strong'", "No'"])
id label: Yes'
outlook label: Yes'
temp label: Yes'
humidity label: Yes'
wind label: Yes'
["No'", "Yes'"]
[0.09375, 0.10495626822157432]
No' :::: Yes'
Number of Correct Predictions : Number of Samples 2 : 3
Accuracy: 66.66666666666667
```

6. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

```
# In[1]:

from sklearn.datasets import fetch_20newsgroups_vectorized
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn import metrics

# In[2]:

doc= fetch_20newsgroups_vectorized()
x_train, x_test, y_train, y_test= train_test_split(doc.data,doc.target)

# In[3]:

model= MultinomialNB()
model.fit(x_train,y_train)

# In[4]:

print("accuracy:")
print(metrics.accuracy_score(y_test, model.predict(x_test)))
print("Precision:")
print(metrics.precision_score(y_test, model.predict(x_test),average=None))
print("Recall:")
print(metrics.recall_score(y_test, model.predict(x_test),average=None))
```

Output:

```
accuracy:
0.7320607988688582
Precision:
[0.89090909 0.86170213 0.84259259 0.3772242  0.92134831 0.87596899
 0.89690722 0.68844221 0.81437126 0.93150685 0.92715232 0.6302521
 0.85321101 0.94285714 0.65405405 0.48923077 0.68604651 0.968
 1.         0.        ]
Recall:
[0.4375     0.51923077 0.59477124 0.848      0.55405405 0.73376623
 0.56129032 0.92567568 0.93150685 0.90066225 0.93333333 0.97402597
 0.62837838 0.84615385 0.98373984 0.95783133 0.90769231 0.83448276
 0.1557377  0.        ]
```

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

```
# In[1]:

import pandas as pd
from urllib.request import urlopen
from pgmpy.models import BayesianModel

# In[2]:

names="A,B,C,D,E,F,G,H,I,J,K,L,M,RESULT"
names=names.split(",")
#st="age,sex,cp,trestbps,chol,fbs,restecg,thalach,exang,oldpeak,slope,ca,thal,num"
#st=st.split(",")

# In[3]:

#data=pd.read_csv("processed.cleveland.csv",names=names)
data=pd.read_csv(urlopen("http://bit.do/heart-disease"),names=names)
data.head()

# In[4]:

model=BayesianModel([("A","B"),("B","C"),("C","RESULT")])
model.fit(data)
#model.fit(data.estimator=MaximumLikelihoodEstimator)

# In[5]:

from pgmpy.inference import VariableElimination
infer=VariableElimination(model)
q=infer.query(variables=["RESULT"],evidence={"A":22})

print(q["RESULT"])
```

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

```
# In[1]:

from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics

# In[2]:

iris = datasets.load_iris()
X_train, X_test,y_train,y_test=train_test_split(iris.data, iris.target)

# In[ ]:

model= KMeans(n_clusters=3)
model.fit(X_train,y_train)

# In[3]:

model.score
print(metrics.accuracy_score(y_test, model.predict(X_test)))

# In[ ]:

from sklearn.mixture import GaussianMixture

# In[ ]:

model2= GaussianMixture(n_components=3)
model2.fit(X_train,y_train)

# In[ ]:

model2.score
print(metrics.accuracy_score(y_test, model2.predict(X_test)))
```

<u>Output</u>:

```
0.9210526315789473
0.34210526315789475
```

9. 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.

```
# Python program to demonstrate    KNN classification algorithm    on IRIS dataset
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split
iris_dataset=load_iris()

print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris_dataset.target_names)
for i in range(len(iris_dataset.target_names)):
        print("\n[{0}]:[{1}]".format(i,iris_dataset.target_names[i]))

print("\n IRIS DATA :\n",iris_dataset["data"])
X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"],
random_state=0)
print("\n Target :\n",iris_dataset["target"])
print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)

x_new = np.array([[5, 2.9, 1, 0.2]])
print("\n XNEW \n",x_new)
prediction = kn.predict(x_new)
print("\n Predicted target value: {}\n".format(prediction))
print("\n Predicted feature name: {}\n".format(iris_dataset["target_names"][prediction]))

i=1
x= X_test[i]
x_new = np.array([x])
print("\n XNEW \n",x_new)

for i in range(len(X_test)):
        x = X_test[i]
        x_new = np.array([x])
        prediction = kn.predict(x_new)
        print("\n Actual : {0} {1},
Predicted :{2}{3}".format(y_test[i],iris_dataset["target_names"][y_test[i]],prediction,iris_da
taset["target_names"][ prediction]))
print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X_test, y_test)))
```

Output:

```
  IRIS FEATURES \ TARGET NAMES:
    ['setosa' 'versicolor' 'virginica']

[0]:[setosa]

[1]:[versicolor]

[2]:[virginica]

  IRIS DATA :
  [[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]
       ……….
  [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]]

  Target :
  [0 0 …….. 0 0 1 1 ……. 1 1 2 2 ……. 2 2]

  X TRAIN
  [[5.9 3.   4.2 1.5]
  [5.8 2.6 4.   1.2]
  [6.8 3.   5.5 2.1]
  [4.7 3.2 1.3 0.2]
     …………
  [6.3 2.9 5.6 1.8]
  [5.8 2.7 4.1 1. ]
  [7.7 3.8 6.7 2.2]
  [4.6 3.2 1.4 0.2]]

  X TEST
  [[5.8 2.8 5.1 2.4]
  [6.   2.2 4.   1. ]
  [5.5 4.2 1.4 0.2]
  [7.3 2.9 6.3 1.8]
     …………
  [6.4 2.8 5.6 2.2]
  [5.2 2.7 3.9 1.4]
```

```
[5.7 3.8 1.7 0.3]
[6.  2.7 5.1 1.6]]

Y TRAIN
[1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0 1
0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0 0
1 0 2 1 2 1 0 2 0 2 0 0 2 0 2 1 1 1 2 2 1 1 0 1 2 2 0 1 1 1 1 0 0 0 2 1 2
0]

Y TEST
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
1]

XNEW
[[5.  2.9 1.  0.2]]

Predicted target value: [0]

Predicted feature name: ['setosa']

XNEW
[[6.  2.2 4.  1. ]]

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

        ……………………………………

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[2]['virginica']

TEST SCORE[ACCURACY]: 0.97
```

10. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

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

x = np.linspace(-5, 5, 1000)
y = np.log(np.abs((x ** 2) - 1) + 0.5)
x = x + np.random.normal(scale=0.05, size=1000)
plt.scatter(x, y, alpha=0.3)


def local_regression(x0, x, y, tau):
    x0 = np.r_[1, x0]
    x = np.c_[np.ones(len(x)), x]
    xw = x.T * radial_kernel(x0, x, tau)
    beta = np.linalg.pinv(xw @ x) @ xw @ y
    return x0 @ beta

def radial_kernel(x0, x, tau):
    return np.exp(np.sum((x - x0) ** 2, axis=1) / (-2 * tau ** 2))

def plot_lr(tau):
    domain = np.linspace(-5, 5, num=300)
    pred = [local_regression(x0, x, y, tau) for x0 in domain]
    plt.scatter(x, y, alpha=0.3)
    plt.plot(domain, pred, color="red")
    return plt

plot_lr(0.03).show()
```

Output: