Ex. No	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	Date		
1				

Aim:

The aim is to implement the Candidate-Elimination algorithm to find all hypotheses that are consistent with a given set of training data examples stored in a CSV file. This involves maintaining and updating the sets of the most specific (S) and the most general (G) hypotheses throughout the learning process.

Algorithm:

Initialization: Start with the most specific hypothesis (S) and the most general hypothesis (G).(S) is initialized to the most specific hypothesis (all elements are the null value).(G) is initialized to the most general hypothesis (all elements are the wildcard '?').

Iterate Through Training Examples:For each positive example, update (S) to be more general to cover the example, and remove inconsistent hypotheses from (G).For each negative example, update (G) to be more specific to exclude the example, and remove inconsistent hypotheses from (S).

Consistent Hypotheses: At the end of the process, (S) and (G) represent the set of hypotheses consistent with the training examples.

Program:

```
import numpy as npimportpandasaspd
data=pd.DataFrame(data=pd.read_csv('enjoysport.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(speciiic_h)
    general_h=[["?"foriinrange(len(specific_h))]foriinrange(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):iftarget[i]=="yes":
            for x in range(len(specific_h)):ifh[x]!=specific_h[x]:
```

```
                    specific_h[x] ='?'general_h[x][x]='?'
                print(specific_h)print(specific_h)
            iftarget[i]=="no":
                for x in range(len(specific_h)):ifh[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)
        indices=[ifori,valinenumerate(general_h)ifval==['?','?', '?','?', '?', '?']]
        foriinindices:
            general_h.remove(['?', '?', '?', '?', '?', '?'])returnspecific_h, general_h
s_final, g_final = learn(concepts, target)print("Final Specific_h:", s_final,
sep="\n")print("FinalGeneral_h:",g_final,sep="\n"
```

Data Set:

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
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

Output:

```
FinalSpecific_h:
['sunny"warm"?"strong"?"?']


FinalGeneral_h:
[['sunny','?','?','?','?','?'],
 ['?','warm','?','?','?','?']]
```

Result:

Thus the program was successfully executed.

Ex. No	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.	Date
2		

Aim:

To implement and demonstrate the working of the decision tree-based ID3 (Iterative Dichotomiser 3) algorithm. This involves building a decision tree from a given dataset and using the constructed tree to classify new samples.

Algorithm:

Select Attribute: Choose the best attribute (A) to split the dataset based on a criterion (e.g., information gain or Gini impurity).

Split Dataset: Split the dataset into subsets based on the values of attribute (A).

Recursive Build: Recursively apply the above steps to each subset until one of the following conditions is met:All instances in the subset belong to the same class.There are no more attributes to split on.Stopping criteria are met (e.g., maximum tree depth reached).

Create Decision Tree: Create a decision tree node that represents the selected attribute (A).

TrainingDataset:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
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
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

TestDataset:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*
T1	Rain	Cool	Normal	Strong
T2	Sunny	Mild	Normal	Strong

Program:

```
import mathimportcsv

defload_csv(filename):
        nes=csv.reader(open(filename,"r"));                                                    li
        dataset = list(lines)headers = dataset.pop(0)returndataset,headers

classNode:
        definit(self,attribute):self.attribute=attributeself.children=[]self.answer=""

defsubtables(data,col,delete):                                                                 di
        c={}
        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)):foryinrange(r):
                if data[y][col]==attr[x]:counts[x]+=1

        forxinrange(len(attr)):
                dic[attr[x]]=[[0 for i in range(c)] for j inrange(counts[x])]
                pos=0
                foryinrange(r):
                        if data[y][col]==attr[x]:ifdelete:
                                del data[y][col]dic[attr[x]][pos]=data[y]pos+=1
```

```
        returnattr,dic
defentropy(S):
        tr=list(set(S))                                                    at
        if len(attr)==1:return0

        counts=[0,0]
        foriinrange(2):
                counts[i]=sum([1forxinSifattr[i]==x])/(len(S)*1.0)

        sums=0
        forcntincounts:
                sums+=-1*cnt*math.log(cnt,2)returnsums

defcompute_gain(data,col):                                                 at
        tr,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])forx inrange(len(attr)):
                ratio[x]=len(dic[attr[x]])/(total_size*1.0)entropies[x]=entropy([row[-1]forrowin
dic[attr[x]]])
                total_entropy-=ratio[x]*entropies[x]returntotal_entropy

defbuild_tree(data,features):                                              la
        stcol=[row[-1]forrowindata]if(len(set(lastcol)))==1:
                node=Node("")node.answer=lastcol[0]returnnode

        n=len(data[0])-1gains=[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))
    returnnode


defprint_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)return

    print("  "*level,node.attribute)forvalue,ninnode.children:
        print("  "*(level+1),value)print_tree(n,level+2)


defclassify(node,x_test,features):
ifnode.answer!="":
        print(node.answer)return
    pos=features.index(node.attribute)forvalue,ninnode.children:
        if x_test[pos]==value:classify(n,x_test,features)

'''Mainprogram'''
dataset,features=load_csv("data3.csv")node1=build_tree(dataset,features)


print("The decision tree for the dataset using ID3 algorithmis")
print_tree(node1,0)testdata,features=load_csv("data3_test.csv")forxtest intestdata:
    print("Thetestinstance:",xtest)
    print("Thelabelfortestinstance:",end=" ")classify(node1,xtest,features)
```

Output:

ThedecisiontreeforthedatasetusingID3algorithmis

[redacted]
rain
[redacted]
strong
[redacted]
weak
[redacted]
overcast
yes

sunny
[redacted]
normal
[redacted]
high
[redacted]

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

Result:

Thus the program was successfully executed.

Ex. No	Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.	Date
3		

Aim:

To implement an Artificial Neural Network (ANN) using the Backpropagation algorithm and test its performance using appropriate datasets. This involves training the neural network on a dataset to learn patterns and then using the trained network to make predictions or classifications on new data.

Algorithm:

Load Dataset: Load a dataset suitable for training and testing the ANN. The dataset should have input features and corresponding target labels.

Initialize Network: Initialize the neural network with random weights.Forward Pass:Implement the forward pass to compute the output of the network given input data.

Compute Loss:Compute the loss or error between the predicted output and the actual output using a suitable loss function (e.g., mean squared error).

Backward Pass:Implement the backward pass to compute the gradients of the loss function with respect to the weights.

Update Weights:Use gradient descent or its variants to update the weights of the network based on the computed gradients.

Training Loop:Iteratively perform forward and backward passes on the training data for a fixed number of epochs.

Test the ANN:After training, evaluate the performance of the ANN on a separate test dataset to assess its generalization ability.

TrainingExamples:

Example	Sleep	Study	Expected %in Exams
1	2	9	92
2	1	5	86
3	3	6	89

Normalizetheinput

Example	Sleep	Study	Expected %inExams
1	2/3=0.66666667	9/9 =1	0.92
2	1/3=0.33333333	5/9=0.55555556	0.86
3	3/3 =1	6/9=0.66666667	0.89

Program:

```
importnumpyasnp
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 longitudinallyy= y/100

#Sigmoid Functiondefsigmoid(x):
    return1/(1+np.exp(-x))

#Derivative of Sigmoid Functiondefderivatives_sigmoid(x):
    returnx*(1- x)

#Variableinitialization
epoch=5000                    #Setting training iterationslr=0.1    #Settinglearningrate
inputlayer_neurons=2          #number of features in data sethiddenlayer_neurons=3
                              #number of hidden layers neuronsoutput_neurons=1
                              #numberofneuronsatoutputlayer
#weight and bias initializationwh=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*yfori in range(epoch):
#Forward Propogationhinp1=np.dot(X,wh)hinp=hinp1+bh
    hlayer_act = sigmoid(hinp)outinp1=np.dot(hlayer_act,wout)outinp=outinp1+bout
```

```
        output=sigmoid(outinp)

#BackpropagationEO=y-output
        outgrad = derivatives_sigmoid(output)d_output=EO* outgrad
        EH=d_output.dot(wout.T)

#how much hidden layer wts contributed to errorhiddengrad =
        derivatives_sigmoid(hlayer_act)d_hiddenlayer=EH *hiddengrad

# dotproduct of nextlayererror and currentlayeropwout+=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("PredictedOutput:\n",output)
```

Output:

```
Input:
[[0.666666671.                    ]
 [0.333333330.55555556]
 [1.                  0.66666667]]

Actual Output:[[0.92]
 [0.86]
 [0.89]]

Predicted Output:[[0.89726759]
 [0.87196896]
[0.9000671]]
```

Result:

Thus the program was successfully executed.

Ex. No	Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file and compute the accuracy with a few test data sets.	Date
4		

Aim:

To implement a Naive Bayes classifier to classify data based on a sample training dataset stored in a CSV file. The Naive Bayes classifier is a simple probabilistic classifier based on Bayes' theorem with the assumption of independence between every pair of features.

Algorithm:

Load Dataset: Load the dataset from a CSV file.

Calculate Prior Probabilities: Calculate the prior probabilities for each class based on the training data.

Calculate Likelihoods: For each feature, calculate the likelihood given each class.

Compute Posterior Probabilities: Use the likelihoods and priors to compute the posterior probabilities.

Predict Class: Classify new samples based on the computed posterior probabilities.

Program:

```
import csvimport randomimportmath
defloadcsv(filename):
        lines = csv.reader(open(filename, "r"));dataset= list(lines)
        foriinrange(len(dataset)):
            #convertingstringsintonumbersforprocessing
        returndataset
defsplitdataset(dataset,splitratio):
        #67%trainingsize
        trainset=[]
        copy=list(dataset);
```

```
#generateindicesforthedatasetlistrandomlytopickelefortrainingdata
        whilelen(trainset)<trainsize:
            index = random.randrange(len(copy));trainset.append(copy.pop(index))
        return[trainset,copy]


defseparatebyclass(dataset):

#createsadictionaryofclasses1and0wherethevaluesare#theinstancesbelongingto eachclass
        separated={}#dictionaryofclasses1and0
        for i in range(len(dataset)):vector=dataset[i]
            if (vector[-1] not in separated):separated[vector[-1]]=[]
            separated[vector[-1]].append(vector)returnseparated


defmean(numbers):
        returnsum(numbers)/float(len(numbers))


defstdev(numbers):
        avg=mean(numbers)
        variance = sum([pow(x-avg,2) for x innumbers])/float(len(numbers)-1)
        returnmath.sqrt(variance)
def summarize(dataset): #creates a dictionary of classessummaries                    =
                        [(mean(attribute),                stdev(attribute))
                        for
attributeinzip(*dataset)];
        del summaries[-1] #excluding labels +ve or -vereturnsummaries


defsummarizebyclass(dataset):
        separated = separatebyclass(dataset);#print(separated)
        summaries={}
        forclassvalue,instancesinseparated.items():
#forkey,valueindic.items()
#summaries[classvalue]=listoftuples(mean,std)foreachclassvalue
```

```
            summaries[classvalue] = summarize(instances)#summarizeisusedtocal
tomeanandstd
        returnsummaries

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

defcalculateclassprobabilities(summaries,inputvector):
#probabilitiescontainstheallprobofallclassoftestdataprobabilities= {}
        forclassvalue,classsummariesinsummaries.items():#classandattributeinformationasmeanand
sd
                probabilities[classvalue]=1
                foriinrange(len(classsummaries)):
                        mean, stdev = classsummaries[i] #take mean
andsdofeveryattributeforclass0and1seperaely
                        x = inputvector[i] #testvector's first attributeprobabilities[classvalue]*=
calculateprobability(x, mean, stdev);#use normal distreturnprobabilities

def predict(summaries, inputvector): #training and test dataispassed
        probabilities = calculateclassprobabilities(summaries,inputvector)
        bestLabel,bestProb=None,-1
        for classvalue, probability in probabilities.items():#assignsthatclasswhichhasthehighestprob
                if bestLabel is None or probability > bestProb:bestProb=probability
                        bestLabel = classvaluereturn bestLabel

def getpredictions(summaries, testset):predictions= []
        foriinrange(len(testset)):
                result = predict(summaries, testset[i])predictions.append(result)
        returnpredictions

def getaccuracy(testset, predictions):correct= 0
```

```
        foriinrange(len(testset)):
                if testset[i][-1] == predictions[i]:correct+= 1
        return(correct/float(len(testset)))*100.0


defmain():
        filename = 'naivedata.csv'splitratio =0.67
        dataset=loadcsv(filename);


        trainingset, testset = splitdataset(dataset,
        splitratio)print('Split{0}rowsintotrain={1}andtest={2}
rows'.format(len(dataset), len(trainingset), len(testset)))#prepare model
        summaries = summarizebyclass(trainingset);#print(summaries)
       #testmodel
        predictions = getpredictions(summaries, testset) #find
thepredictionsoftestdatawiththetraining data
        accuracy = getaccuracy(testset, predictions)print('Accuracyoftheclassifieris:
{0}%'.format(accuracy))main()
```

Output:

```
Split768rowsintotrain=514
andtest=254rowsAccuracyoftheclassifieris:71.65354330708661%
```

Result:

Thus the program was successfully executed.

Ex. No	Implement naïve Bayesian Classifier model to classify a set of documents and measure the accuracy, precision, and recall.	Date
5		

Aim:

To implement a Naive Bayes Classifier to classify a set of documents and measure its performance using metrics such as accuracy, precision, and recall. Naive Bayes is a probabilistic classifier based on Bayes' theorem with the assumption of independence between features.

Algorithm:

Preprocessing:Convert the documents into a suitable format (e.g., bag-of-words model).Split the dataset into training and testing sets.

Calculate Prior Probabilities:Compute the prior probability of each class based on the training data.

Calculate Likelihood:For each feature, calculate the likelihood of that feature given a class.

Apply Bayes' Theorem:Use Bayes' theorem to compute the posterior probability of each class given a document.

Classify:Assign the class with the highest posterior probability to the document.

Evaluate Performance:Measure the accuracy, precision, and recall of the classifier using the test data.

Dataset:

	TextDocuments	Label
1	Ilovethissandwich	pos
2	Thisisanamazing place	pos
3	Ifeelverygoodaboutthesebeers	pos
4	Thisismy bestwork	pos
5	Whatanawesomeview	pos
6	Idonotlikethisrestaurant	neg
7	Iamtiredofthisstuff	neg
8	Ican'tdealwiththis	neg
9	Heismyswornenemy	neg

10	Mybossishorrible	neg
11	Thisisanawesome place	pos
12	Idonotlikethetasteofthis juice	neg
13	Ilove todance	pos
14	Iamsickandtired ofthisplace	neg
15	Whatagreatholiday	pos
16	Thatisabadlocalityto stay	neg
17	Wewillhavegood fun tomorrow	pos
18	Iwenttomyenemy'shousetoday	neg

Program:

```
import pandas as

pdmsg=pd.read_csv('naivetext.csv',names=['message','label'])print('Thedimensionsofthedataset',msg.s

hape)

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


print(X)print(y)

#splittingthedatasetintotrainandtestdata
from sklearn.model_selection import train_test_splitxtrain,xtest,ytrain,ytest=train_test_split(X,y)


print ('\n The total number of Training Data
:',ytrain.shape)print('\nThetotalnumberofTestData:',ytest.shape)


#outputofcountvectoriserisasparsematrix
from sklearn.feature_extraction.text import CountVectorizercount_vect= CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)xtest_dtm=count_vect.transform(xtest)
print('\n The words or Tokens in the text documents \n')print(count_vect.get_feature_names())
```

```
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())

#TrainingNaiveBayes(NB)classifierontrainingdata.
fromsklearn.naive_bayesimportMultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)predicted=clf.predict(xtest_dtm)

#printingaccuracy,Confusionmatrix,PrecisionandRecall
 fromsklearn import metrics
print('\nAccuracyoftheclassiferis',
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))
```

Output:

```
Thedimensionsof the dataset(18,2)
0     Ilovethissandwich
1     Thisisanamazing place
2     Ifeelverygoodaboutthesebeers
3     Thisismy bestwork
4     Whatanawesomeview
5     Idonotlikethisrestaurant
6     Iamtiredofthisstuff
7     Ican'tdealwiththis
8     Heismyswornenemy
9     Mybossishorrible
10    Thisisanawesome place
11    Idonotlikethe tasteofthisjuice
12    Ilove todance
13    Iamsickandtiredofthisplace
14    Whatagreatholiday
15    Thatisabadlocalityto stay
16    Wewillhavegood fun tomorrow
17    Iwenttomyenemy'shousetoday
```

Name: message, dtype: object0 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

The total number of Training Data: (13,)Thetotalnumberof TestData:(5,)

Thewords orTokensinthetextdocuments

['about','am','amazing','an','and','awesome','beers','best','can','deal','do','enemy','feel',

'fun','good','great','have','he','holiday','house','is','like','love','my','not','of','place',

'restaurant','sandwich','sick','sworn','these','this','tired','to','today','tomorrow','very','view','we','went','what','will','with','work']

Accuracy of the classifier is 0.8Confusionmatrix

[[21]

[02]]

The value of Precision 0.6666666666666666ThevalueofRecall1.0

Result:

Thus the program was successfully executed.

Ex. No	Write a program to construct a Bayesian network to diagnose CORONA infection using standard WHO Data Set.	Date		
6				

Aim:

To construct a Bayesian Network to diagnose COVID-19 infection using a standard dataset. A Bayesian Network is a graphical model that represents the probabilistic relationships among a set of variables. In the context of COVID-19 diagnosis, it can be used to model the relationships between various symptoms and the likelihood of infection.

Algorithm:

Define Variables: Identify and define the variables involved, such as symptoms (e.g., fever, cough, fatigue), test results, and the infection status.

Structure the Network: Define the structure of the Bayesian Network by determining which variables are conditionally dependent on each other.

Parameter Learning: Estimate the conditional probability tables for each variable given its parents in the network. This can be done using a dataset.

Inference: Use the Bayesian Network to perform inference, i.e., compute the probability of infection given observed symptoms and test results.

Program:

```
import numpy as npimport pandas as pdimportcsv
frompgmpy.estimatorsimportMaximumLikelihoodEstimatorfrompgmpy.modelsimportBa
yesianModel
frompgmpy.inferenceimportVariableElimination


#readClevelandHeartDiseasedata
heartDisease=pd.read_csv('heart.csv')
heartDisease=heartDisease.replace('?',np.nan)

#displaythedata
print('Sample instances from the dataset are given below')print(heartDisease.head())

#displaytheAttributesnamesanddatatyes

print('\n Attributes and datatypes')print(heartDisease.dtypes)
```

```
#CreatModel-BayesianNetwork
model =BayesianModel([('age','heartdisease'),('sex','heartdisease'),(
'exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])

#LearningCPDsusingMaximumLikelihoodEstimators

print('\n Learning CPD using Maximum likelihood
estimators')model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

#InferencingwithBayesianNetwork

print('\n Inferencing with Bayesian
Network:')HeartDiseasetest_infer=VariableElimination(model)

#computingtheProbabilityofHeartDiseasegivenrestecg

print('\n1.ProbabilityofHeartDiseasegivenevidence=restecg
:1')q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})

print(q1)

#computingtheProbabilityofHeartDiseasegivencp

print('\n2.ProbabilityofHeartDiseasegivenevidence=cp:2')q2=HeartDiseasetest_infer.query(vari
ables=['heartdisease'],evidence={'cp':2})

print(q2)
```

Output:

```
================ RESTART: E:\ML Lab - 2020-21\MLLab-7\ML7.py ================
Few examples from the dataset are given below
   age  sex  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease
0   63    1   1       145   233  ...      2.3      3   0     6             0
1   67    1   4       160   286  ...      1.5      2   3     3             2
2   67    1   4       120   229  ...      2.6      2   2     7             1
3   37    1   3       130   250  ...      3.5      3   0     3             0
4   41    0   2       130   204  ...      1.4      1   0     3             0

[5 rows x 14 columns]

 Attributes and datatypes
age             int64
sex             int64
cp              int64
trestbps        int64
chol            int64
fbs             int64
restecg         int64
thalach         int64
exang           int64
oldpeak       float64
slope           int64
ca             object
thal           object
heartdisease    int64
dtype: object
```

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence= restecg

heartdisease	phi(heartdisease)
heartdisease(0)	0.1012
heartdisease(1)	0.0000
heartdisease(2)	0.2392
heartdisease(3)	0.2015
heartdisease(4)	0.4581

2. Probability of HeartDisease given evidence= cp

heartdisease	phi(heartdisease)
heartdisease(0)	0.3610
heartdisease(1)	0.2159
heartdisease(2)	0.1373
heartdisease(3)	0.1537
heartdisease(4)	0.1321

Result:

Thus the program was successfully executed.

Ex. No	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.	Date		
7				

Aim:

To apply the Expectation-Maximization (EM) algorithm and the k-Means algorithm to cluster a set of data stored in a CSV file, and then compare the results of these two algorithms.

Algorithm:

Load Dataset: Load the data from a CSV file.

Preprocess Data: Standardize or normalize the data if necessary.

Apply k-Means Algorithm:Initialize (k) cluster centroids.Assign each data point to the nearest centroid.Update centroids based on the mean of assigned points.Repeat until convergence.

Apply EM Algorithm (GMM):Initialize parameters (means, covariances, and mixture weights).E-step: Calculate the responsibilities for each data point.

M-step: Update parameters based on the responsibilities.Repeat until convergence.Compare Results: Use metrics such as silhouette score, inertia, or log-likelihood to compare the results.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=100, centers =
4,Cluster_std=0.60,random_state=0)
X = X[:, ::-1]
```

#flip axes for better plotting

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture (n_components = 4).fit(X)
lables = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap="viridis");
probs = gmm.predict_proba(X)
print(probs[:5].round(3))
```

```
size = 50 * probs.max(1) ** 2 # square emphasizes differences
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap="viridis", s=size);

from matplotlib.patches import Ellipse
def draw_ellipse(position, covariance, ax=None, **kwargs);
"""Draw an ellipse with a given position and covariance"""
Ax = ax or plt.gca()
# Convert covariance to principal axes
if covariance.shape ==(2,2):

U, s, Vt = np.linalg.svd(covariance)
Angle = np.degrees(np.arctan2(U[1, 0], U[0,0]))
Width, height = 2 * np.sqrt(s)
else:
angle = 0
width, height = 2 * np.sqrt(covariance)

#Draw the Ellipse
for nsig in range(1,4):
ax.add_patch(Ellipse(position, nsig * width, nsig *height,
angle, **kwargs))
def plot_gmm(gmm, X, label=True, ax=None):
ax = ax or plt.gca()
labels = gmm.fit(X).predict(X)

    if label:

ax.scatter(X[:, 0], x[:, 1], c=labels, s=40, cmap="viridis", zorder=2)
else:
ax.scatter(X[:, 0], x[:, 1], s=40, zorder=2)
ax.axis("equal")

w_factor = 0.2 / gmm.weights_.max()
for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
```

```
draw_ellipse(pos, covar, alpha=w * w_factor)

gmm = GaussianMixture(n_components=4, random_state=42)
plot_gmm(gmm, X)
gmm = GaussianMixture(n_components=4, covariance_type="full",
random_state=42)

    plot_gmm(gmm, X)
```

K-means

```
from sklearn.cluster import KMeans

#from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=pd.read_csv("kmeansdata.csv")
df1=pd.DataFrame(data)
print(df1)
f1 = df1['Distance_Feature'].values
f2 = df1['Speeding_Feature'].values

X=np.matrix(list(zip(f1,f2)))
plt.plot()
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.ylabel('speeding_feature')
plt.xlabel('Distance_Feature')
plt.scatter(f1,f2)
plt.show()

# create new plot and data
plt.plot()
```

```
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# KMeans algorithm
#K = 3
kmeans_model = KMeans(n_clusters=3).fit(X)

plt.plot()
for i, l in enumerate(kmeans_model.labels_):
plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.show()
```

Driver_ID,Distance_Feature,Speeding_Feature

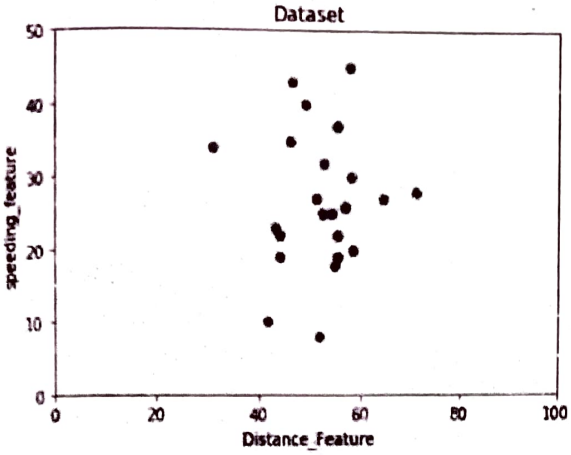
3423311935,71.24,28
3423313212,52.53,25
3423313724,64.54,27
3423311373,55.69,22
3423310999,54.58,25
3423313857,41.91,10
3423312432,58.64,20
3423311434,52.02,8
3423311328,31.25,34
3423312488,44.31,19
3423311254,49.35,40
3423312943,58.07,45
3423312536,44.22,22
3423311542,55.73,19
3423312176,46.63,43
3423314176,52.97,32
3423314202,46.25,35
3423311346,51.55,27
3423310666,57.05,26

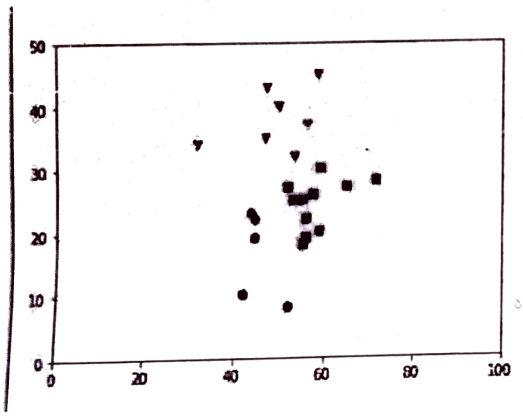
3423313527,58.45,30

3423312182,43.42,23

3423313590,55.68,37

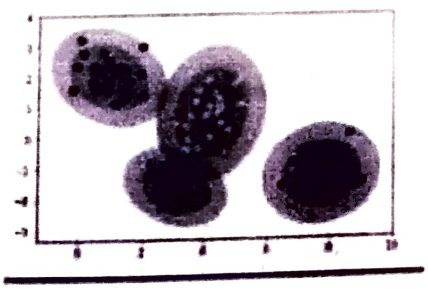
3423312268,55.15,18





Output

[[1 ,0, 0, 0]
[0 ,0, 1, 0]
[1 ,0, 0, 0]
[1 ,0, 0, 0]
[1 ,0, 0, 0]]



Result:

Thus the program was successfully executed.

Ex. No	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.	Date		
8				

Aim:

To implement the k-Nearest Neighbors (k-NN) algorithm to classify the Iris dataset and print both correct and wrong predictions. The k-NN algorithm is a simple, instance-based learning method used for classification and regression tasks.

Algorithm:

Load and Preprocess Data: Load the dataset and preprocess it (e.g., normalize features).

Distance Calculation: Compute the distance between the test instance and all training instances.

Find Neighbors: Identify the k-nearest neighbors to the test instance.

Majority Voting: Assign the class label based on the majority class among the k-nearest neighbors.

Classification: Repeat the process for all test instances to classify them.

Evaluation: Compare the predicted labels with the actual labels and print both correct and incorrect predictions.

Program:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
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']
model = KMeans(n_clusters=3)
model.fit(X)
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 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
plt.subplot(1, 2, 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_))

from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)
from sklearn.mixture import GaussianMixture
```

```
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_gmm = gmm.predict(xs)

#y_cluster_gmm
plt.subplot(2, 2, 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))
```

Result:

Thus the program was successfully executed.