Ex.No. 1 **Date :**

Implementation of FINDS Algorithm

Aim

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

Procedure

Step 1: Initialize h to the most specific hypothesis in H

Step 2: For each positive training instance x

For each attribute constraint a_i in h

If the constraint a_i is satisfied by x

then do nothing

Else

replace ai in h by the next more general constraint that is satisfied by x

Step 3: Output the hypothesis h

Dataset

sky	**temp**	**humidity**	**wind**	**water**	**forecast**	**target**
sunny	warm	normal	strong	warm	same	True
sunny	warm	high	strong	warm	same	True
rainy	cold	high	strong	warm	change	False
sunny	warm	high	strong	cool	change	True

Program

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

Output

```
for the training sample 0
the hypothesis is: [['0', '0', '0', '0', '0', '0']]
for the training sample 1
the hypothesis is: [['sunny', 'warm', 'normal', 'strong', 'warm', 'same']]
for the training sample 2
the hypothesis is: [['sunny', 'warm', '?', 'strong', 'warm', 'same']]
for the training sample 3
the hypothesis is: [['sunny', 'warm', '?', 'strong', 'warm', 'same']]
for the training sample 4
the hypothesis is: [['sunny', 'warm', '?', 'strong', '?', '?']]
the maximally specific hypothesis is [['sunny', 'warm', '?', 'strong', '?', '?']]
```

Result

FINDS algorithm has been implemented and demonstrated with a sample dataset

Ex.No. 2 **Date :**

Implementation of Candidate Elimination Algorithm

Aim

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

Procedure

G ←maximally general hypotheses in H

S ←maximally specific hypotheses in H

For each training example d, do:

Case 1: If d is positive example

Remove from G any hypothesis h inconsistent with d

For each hypothesis s in S not consistent with d:

Remove s from S

Add to S all minimal generalizations of s consistent with d and having a generalization in G

Remove from S any hypothesis with a more specific h in S

Case 2: If d is negative example

Remove from S any hypothesis h inconsistent with d

For each hypothesis g in G not consistent with d:

Remove g from G

Add to G all minimal specializations of g consistent with d and having a specialization in S

Remove from G any hypothesis having a more general hypothesis in G

Dataset

Sky	temp	humidity	wind	water	forecast	target
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

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

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

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

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

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

```
        print("\n")

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

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

Output

```
Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values are:  ['yes' 'yes' 'no' 'yes']
Initialization of specific_h and genearal_h
Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Instance 3 is ['rainy' 'cold' 'high' 'strong' 'warm' 'change']

Instance is Negative

Specific Bundary after 3 Instance is ['sunny' 'warm' '?' 'strong' 'warm' 'same']

Generic Boundary after 3 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

Instance 4 is ['sunny' 'warm' 'high' 'strong' 'cool' 'change']

Instance is Positive

Specific Bundary after 4 Instance is ['sunny' 'warm' '?' 'strong' '?' '?']

Generic Boundary after 4 Instance is [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:

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

Final General_h:

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

<u>Result</u>

Candidate Elimination algorithm has been implemented and demonstrated with sample dataset.

Ex.No. 3 **Date :**

Implementation of Decision Tree Algorithm

Aim

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

Procedure

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples.

Target_attribute is the attribute whose value is to be predicted by the tree.

Attributes is a list of other attributes that may be tested by the learned decision tree.

Returns a decision tree that correctly classifies the given Examples.

Create a Root node for the tree

If all Examples are positive, Return the single-node tree Root, with label = +

If all Examples are negative, Return the single-node tree Root, with label = -

If Attributes is empty, Return the single-node tree Root,

with label = most common value of Target_attribute in Examples

Otherwise Begin

A ← the attribute from Attributes that best* classifies Examples

The decision attribute for Root ← A

For each possible value, vi, of A,

    Add a new tree branch below Root, corresponding to the test A = vi

    Let Examples vi, be the subset of Examples that have value vi for A

    If Examples vi , is empty

        Then below this new branch add a leaf node with

        label = most common value of Target_attribute in Examples

    Else

        below this new branch add the subtree

    ID3(Examples vi, Targe_tattribute, Attributes – {A}))

    End

Return Root

Program

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

data = pd.read_csv("play.csv")
features = [feat for feat in data]
features.remove("answer")

class Node:
  def __init__(self):
    self.children = []
    self.value = ""
    self.isLeaf = False
    self.pred = ""
def entropy(examples):
  pos = 0.0
  neg = 0.0
  for _, row in examples.iterrows():
    if row["answer"] == "yes":
      pos += 1
    else:
      neg += 1
  if pos == 0.0 or neg == 0.0:
    return 0.0
  else:
    p = pos / (pos + neg)
    n = neg / (pos + neg)
    return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
  uniq = np.unique(examples[attr])
  #print ("\n",uniq)
  gain = entropy(examples)
  #print ("\n",gain)
```

```
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain

def ID3(examples, attrs):
    root = Node()
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
```

```
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:", child.pred)
                exit
            else:
                classify (child.children[0], new)
####
root = ID3(data, features)
print("Decision Tree is:")
printTree(root)
print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}
classify (root, new)
```

Output

Decision Tree is:

outlook

overcast -> ['yes']

rainy -> ['no']

sunny -> ['yes']

Predicted Label for new example {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'normal', 'wind': 'strong'} is: ['yes']

Result

ID3 based Decision Tree algorithm has been implemented and tested with sample dataset

Ex.No. 4 **Date :**

Building Artificial Neural Network by implementing Backpropagation Algorithm

Aim

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

Procedure

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

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

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

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

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

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

#draws a random range of numbers uniformly of dim x*y
```

```
for i in range(epoch):
   #Forward Propogation
   hinp1=np.dot(X,wh)
   hinp=hinp1 + bh
   hlayer_act = sigmoid(hinp)
   outinp1=np.dot(hlayer_act,wout)
   outinp= outinp1+bout
   output = sigmoid(outinp)

   #Backpropagation
   EO = y-output
   outgrad = derivatives_sigmoid(output)
   d_output = EO * outgrad
   EH = d_output.dot(wout.T)
   hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to
error
   d_hiddenlayer = EH * hiddengrad

   wout += hlayer_act.T.dot(d_output) *lr   # dotproduct of nextlayererror and
currentlayerop
   wh += X.T.dot(d_hiddenlayer) *lr
   print ("-----------Epoch-", i+1, "Starts----------")
   print("Input: \n" + str(X))
   print("Actual Output: \n" + str(y))
   print("Predicted Output: \n" ,output)
   print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output

```
-----------Epoch- 1 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
```

```
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.757371  ]
 [0.7404456 ]
 [0.74680043]]
-----------Epoch- 1 Ends----------

-----------Epoch- 2 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.75999901]
 [0.74289718]
 [0.74936496]]
-----------Epoch- 2 Ends----------

-----------Epoch- 3 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
```

```
Predicted Output:
 [[0.76254479]
 [0.74527475]
 [0.75185111]]
-----------Epoch- 3 Ends----------

-----------Epoch- 4 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.76501199]
 [0.74758151]
 [0.7542623 ]]
-----------Epoch- 4 Ends----------

-----------Epoch- 5 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.76740405]
 [0.74982049]
 [0.75660176]]
-----------Epoch- 5 Ends----------
```

Input:

```
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
```

Actual Output:

```
[[0.92]
 [0.86]
 [0.89]]
```

Predicted Output:

```
 [[0.76740405]
 [0.74982049]
 [0.75660176]]
```

Result

Built the Artificial Neural Network by implementing the Backpropagation algorithm and tested the same using appropriate data sets

Ex.No. 5 **Date :**

Implementation of naive Bayesian classifier -I

Aim

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.

Procedure

Step 1: Calculate the prior probability for given class labels
Step 2: Find Likelihood probability with each attribute for each class
Step 3: Put these value in Bayes Formula and calculate posterior probability.
Step 4: See which class has a higher probability, given the input belongs to the higher probability class

Dataset

Examples	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	121	33.6	0.627	50	1
2	1	85	66	29	110	26.6	0.351	31	0
3	8	183	64	31	212	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	2	137	40	35	168	43.1	2.288	33	1
6	5	116	74	31	156	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	0
8	10	115	47	31	167	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1

Program

```
import csv
import random
import math

def loadcsv(filename):
        lines = csv.reader(open(filename, "r"));
        dataset = list(lines)
        for i in range(len(dataset)):
    #converting strings into numbers for processing
                dataset[i] = [float(x) for x in dataset[i]]

        return dataset

def splitdataset(dataset, splitratio):
  #67% training size
        trainsize = int(len(dataset) * splitratio);
        trainset = []
        copy = list(dataset);
        while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for training data
                index = random.randrange(len(copy));
                trainset.append(copy.pop(index))
        return [trainset, copy]

def separatebyclass(dataset):
        separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
        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): #creates a dictionary of classes
	summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
	del summaries[-1] #excluding labels +ve or -ve
	return summaries

def summarizebyclass(dataset):
	separated = separatebyclass(dataset);
   #print(separated)
	summaries = {}
	for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
		summaries[classvalue] = summarize(instances) #summarize is used to cal to
mean and std
	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 = {} # probabilities contains the all prob of all class of test data
	for classvalue, classsummaries in summaries.items():#class and attribute information
as mean and sd
		probabilities[classvalue] = 1
		for i in range(len(classsummaries)):
			mean, stdev = classsummaries[i] #take mean and sd of every attribute
for class 0 and 1 seperaely
			x = inputvector[i] #testvector's first attribute
			probabilities[classvalue] *= calculateprobability(x, mean, stdev);#use
normal dist
	return probabilities

def predict(summaries, inputvector): #training and test data is passed
	probabilities = calculateclassprobabilities(summaries, inputvector)
	bestLabel, bestProb = None, -1
	for classvalue, probability in probabilities.items():#assigns that class which has he
highest prob
		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 = '5-dataset.csv'
    splitratio = 0.67
    dataset = loadcsv(filename);

    trainingset, testset = splitdataset(dataset, splitratio)
    print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset),
len(trainingset), len(testset)))
    # prepare model
    summaries = summarizebyclass(trainingset);
    #print(summaries)
  # test model
    predictions = getpredictions(summaries, testset) #find the predictions of test data
with the training data
    accuracy = getaccuracy(testset, predictions)
    print('Accuracy of the classifier is : {0}%'.format(accuracy))
 main()
```

Output

Split 767 rows into train=513 and test=254 rows
Accuracy of the classifier is : 79.13%

Result

Naive Bayesian Classifier-I has been implemented and tested with sample dataset

Ex.No. 6 **Date :**

Implementation of naive Bayesian classifier -II

Aim

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

Procedure

Step 1: Calculate the prior probability for given class labels
Step 2: Find Likelihood probability with each attribute for each class
Step 3: Put these value in Bayes Formula and calculate posterior probability.
Step 4: See which class has a higher probability, given the input belongs to the higher probability class

Program

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

msg=pd.read_csv('6-Dataset.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

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

#splitting the dataset into train and test data
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)


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

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

```
#printing accuracy, Confusion matrix, Precision and Recall
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\n The value of Precision', metrics.precision_score(ytest,predicted))
print('\n The value of Recall', metrics.recall_score(ytest,predicted))
```

Output

```
The dimensions of the dataset (18, 2)

 the total number of Training Data : (13,)

 the total number of Test Data : (5,)

 The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'bad', 'beers', 'boss', 'dance', 'do', 'enemy',
'feel', 'fun', 'good', 'great', 'have', 'he', 'holiday', 'horrible', 'house', 'is', 'juice', 'like', 'locality',
'love', 'my', 'not', 'of', 'place', 'sick', 'stay', 'stuff', 'sworn', 'taste', 'that', 'the', 'these', 'this',
'to', 'today', 'tomorrow', 'tried', 'very', 'view', 'we', 'went', 'what', 'will']

 Accuracy of the classifier is 0.8

 Confusion matrix
[[2 0]
 [1 2]]

 The value of Precision 1.0

 The value of Recall 0.6666666666666666
```

Result

Naive Bayesian Classifier-II has been implemented and tested with sample dataset

Ex.No. 7 **Date :**

Demonstration of Bayesian network

Aim:

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

Procedure

Step 1: Get the sample Heart Disease dataset.

Step 2:Find the Bayesian Model using Maximum Likelihood estimators.

Step 3: Find the attributes using the Bayesian Network.

Step 4: Calulate the probality of heart disease from the given evidence

Program

```
import numpy as np
import pandas as pd
#import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('7-dataset.csv')
heartDisease = heartDisease.replace('?',np.nan)

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

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

model=
BayesianModel([('age','heartdisease'),('gender','heartdisease'),('exang','heartdisease'),('cp','
heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

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

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)

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

Output

```
Sample instances from the dataset are given below
  age  gender  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
gender          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
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/4 [00:00<?, ?it/s]
+-----------------+---------------------+
| 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
 0%|          | 0/3 [00:00<?, ?it/s]
 0%|          | 0/3 [00:00<?, ?it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

Result

Demonstration of Bayesian Network has been implemented and tested with sample dataset

Exp. No. 8. **Date:**

Application of EM algorithm for clustering

Aim

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 and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Procedure

Step 1: To cluster a set of data stored in a csv file by using EM algorithm.

Step 2: Plot the data with the two Gaussian distribution.

Step 3: Use the same dataset for clustering using k-means algorithm.

Step 4: Compare the two algorithms and analyse the quality of clustering.

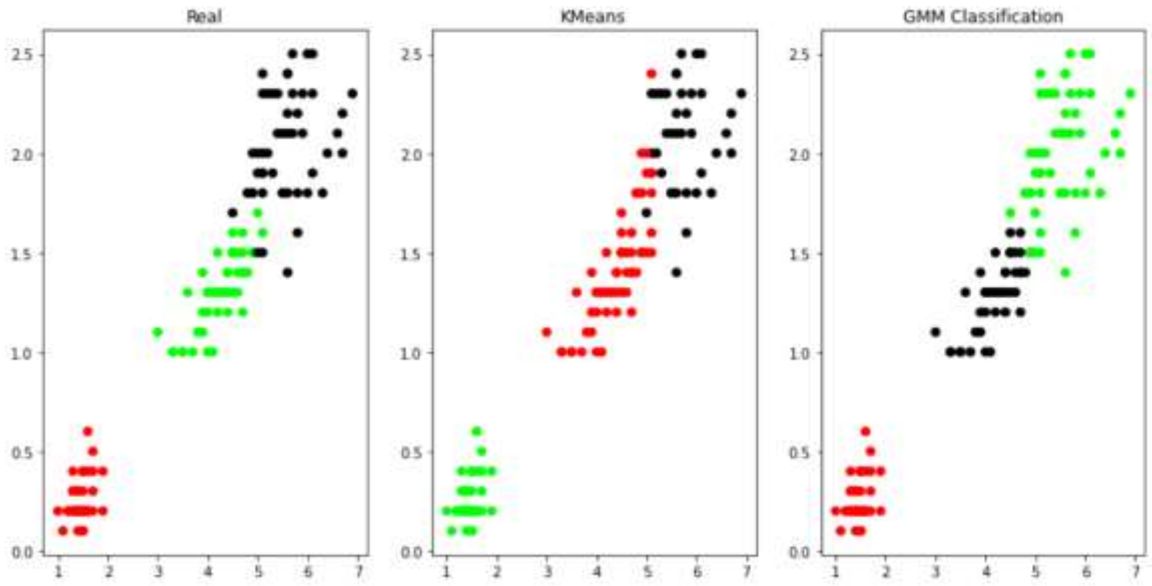
Program

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


names = [‘Sepal_Length’, ‘Sepal_Width’, ‘Petal_Length’, ‘Petal_Width’, ‘Class’]

dataset = pd.read_csv(“8-dataset.csv”, names=names)

X = dataset.iloc[:, :-1]

label = {‘Iris-setosa’:0, ’Iris-versicolor’: 1, ‘Iris-virginica’: 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))
colormap=np.array([‘red’, ‘lime’, ‘black’])

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

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

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

# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM:',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

Output

```
The accuracy score of K-Mean: 0.24
The Confusion matrixof K-Mean:
[[ 0 50 0]
[48 0 2]
[14 0 36]]

The accuracy score of EM: 0.3666666666666664
The Confusion matrix of EM:
[[50 0 0]
[ 0 5 45]
[ 0 50 0]]
```



Result

Application of EM algorithm for clustering has been implemented and tested with sample dataset.

Exp. No. 9. **Date:**

Implementation of k nearest neighbors algorithm for classification

Aim

Write a program to implement the k-Nearest Neighbour algorithm to classify their is data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Procedure

Step1: Read dataset to the pandas data frame.

Step2: Classify using K neighbors classification algorithm for the prediction.

Step3: Confusion matrix for original label, predicted label are found.

Step4: Classification report and accuracy of the classifier is found.

Program

```
import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier


from sklearn.model_selection import train_test_split

from sklearn import metrics

names = [‘sepal-length‘, ‘sepal-width’, ’petal-length’, ‘petal-width&’, ‘Class’]

# Read dataset to pandas dataframe

dataset = pd.read_csv(“9-dataset.csv”, names=names)

X = dataset.iloc[:, :-1]

y = dataset.iloc[:, -1]

print(X.head())

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)
```

```
i = 0

print (&quot;\n----------------------------------------------------------------------&quot;)

print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))

print (&quot;----------------------------------------------------------------------&quot;)

for label in ytest:

print ('%-25s %-25s' % (label, ypred[i]), end="")

if (label == ypred[i]):

print ('%-25s' % ('Correct'))

else:

print (' %-25s'  % ('Wrong'))

i = i + 1

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

print("\nConfusion Matrix:\n", metrics.confusion_matrix(ytest, ypred))

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

print("\nClassification Report:\n" ,metrics.classification_report(ytest, ypred))

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

print('Accuracy of the classifer is %0.2f ', % metrics.accuracy_score(ytest,ypred))

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

<u>Output</u>

```
sepal-length sepal-width petal-length petal-width
0 5.1 3.5 1.4 0.2
1 4.9 3.0 1.4 0.2
2 4.7 3.2 1.3 0.2
3 4.6 3.1 1.5 0.2
4 5.0 3.6 1.4 0.2
----------------------------------------------------------------------
Original Label    Predicted Label   Correct/Wrong
----------------------------------------------------------------------
  Iris-virginica      Iris-virginica        Correct
```

Iris-setosa	Iris-setosa	Correct
Iris-setosa	Iris-setosa	Correct
Iris-virginica	Iris-virginica	Correct
Iris-versicolor	Iris-versicolor	Correct
Iris-versicolor	Iris-virginica	Wrong
Iris-virginica	Iris-virginica	Correct
Iris-versicolor	Iris-versicolor	Correct
Iris-setosa	Iris-setosa	Correct
Iris-versicolor	Iris-versicolor	Correct
Iris-versicolor	Iris-versicolor	Correct
Iris-versicolor	Iris-versicolor	Correct
Iris-setosa	Iris-setosa	Correct
Iris-virginica	Iris-virginica	Correct
Iris-versicolor	Iris-virginica	Wrong

Confusion Matrix:

[[4 0 0]

[0 5 2]

[0 0 4]]

Classification Report:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 4

Iris-versicolor 1.00 0.71 0.83 7

Iris-virginica 0.67 1.00 0.80 4

accuracy 0.87 15

macro avg 0.89 0.90 0.88 15

weighted avg 0.91 0.87 0.87 15

Accuracy of the classifer is 0.87

Result

Implementation of k-nearest neighbors algorithm for classification has been implemented and tested with sample dataset.

Exp. No. 10. **Date:**

Implementation of non-parametric locally weighted regression algorithm

Aim

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

Procedure

Step 1: Read the Given data Sample to X and the curve (linear or non linear) to Y

Step 2: Set the value for Smoothening parameter or Free parameter say τ

Step 3: Set the bias /Point of interest set x0 which is a subset of X

Step 4: Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

Step 5: Determine the value of model term parameter β using:

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

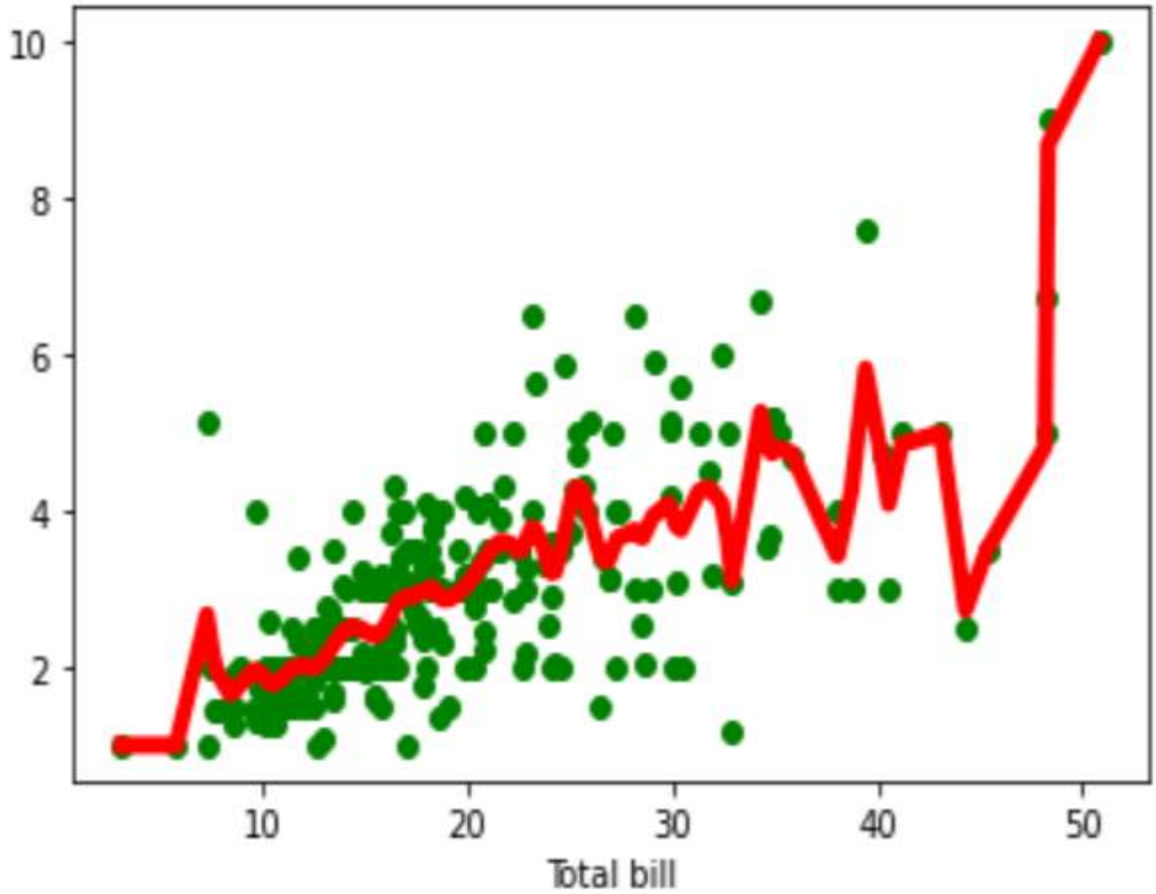
Step 6: Prediction = x0*β

Program

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

def kernel(point, xmat, k):
m,n = np.shape(xmat)
weights = np.mat(np.eye((m)))
for j in range(m):


diff = point - X[j]
weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
return weights

def localWeight(point, xmat, ymat, k):
wei = kernel(point,xmat,k)
W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
```

```
return W

def localWeightRegression(xmat, ymat, k):
m,n = np.shape(xmat)
ypred = np.zeros(m)
for i in range(m):
ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
return ypred

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


#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()


ax = fig.add_subplot(1,1,1)

ax.scatter(bill,tip, color='green')

ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)

plt.xlabel('Total bill')

plt.ylabel('Tip')

plt.show();
```

Output



Result

Implementation of non-parametric locally weighted regression algorithm has been implemented and tested with sample dataset.