DHAANISH AHMED COLLEGE OF ENGINEERING

Dhaanish Nagar, Padappai, Chennai – 601301
Approved By AICTE, New Delhi,
Affiliated to Anna University, Chennai.
www.dhaanish.in

Name : .....

Dept. / Year / Sec :.....

Subject Code & Name :.....

Register No :.....

DHAANISH AHMED COLLEGE OF ENGINEERING

Dhaanish Nagar, Padappai, Chennai – 601301

BONAFIDE CERTIFICATE

This is to certify that, this a Bonafide Record Work done by

Mr./Ms.....

Year.....Semester.....Register No.....

Department of .....in the

.....during the

academic year 20 - 20

_____ _____

Signature

Lab-In-Charge

Signature

Head of the Department

Submitted for the Anna University practical Examination held on.....

Signature of
Internal Examiner
with Date

Signature of
External Examiner
with Date

INDEX

EX. NO	DATE	NAME OF THE EXPERIMENT	PAGE NO.	MARKS	STAFF SIGN
1		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.			
2.		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.			
3		Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.			
4		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.			
5		Implement naïve Bayesian Classifier model to classify a set of documents and measure the accuracy, precision, and recall.			
6		Write a program to construct a Bayesian network to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.			
7		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**.**			
8		Write a program to implement k-Nearest neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.			
9		Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select an appropriate data set for your experiment and draw graphs.			

EXP. NO: 1	**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:	

Aim:

To write a python program to implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples for a given set of training data examples stored in a .CSV file.

Algorithm:

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

Initialize G to the set of maximally general hypotheses in H Initialize S to the set of maximally specific hypotheses in H For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S

- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

Training Samples:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('C:\\Users\\admin\\Downloads\\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(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):
     if target[i] == "yes":
         for x in range(len(specific_h)):
             if h[x]!= specific_h[x]:
                 specific_h[x] ='?'
                 general_h[x][x] ='?'
             print(specific_h)
     print(specific_h)
     if target[i] == "no":
```

```
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)
 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:

```
[['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']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
```

['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
['sunny' 'warm' '?' 'strong' '?' '?']

steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Result:

Thus the Candidate-Elimination algorithm using python program is implemented, executed and output was verified successfully.

EXP. NO: 2	**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:	

Aim:

To write a python program to demonstrate the working of the decision tree based ID3 algorithm by using an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Algorithm:

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to bepredicted 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 valueof 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, v_i, of A,
    - Add a new tree branch below Root, corresponding to the test A = v_i
    - Let Examples v_i, be the subset of Examples that have value v_i for A
    - If Examples v_i , is empty
      - Then below this new branch add a leaf node with label = most commonvalue of Target_attribute in Examples
      - Else below this new branch add the subtree ID3(Examples v_i, Targe_tattribute, Attributes – {A}))
- End
- Return Root
- The best attribute is the one with highest information gain

Entropy:

Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

Where, p+ is the proportion of positive examples in S

p- is the proportion of negative examples in S.

Information Gain:

- **Information gain,** is the expected reduction in entropy caused by partitioning theexamples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examplesS, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Training Dataset:

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

Test Dataset:

Day	Outlook	Temperature	Humidity	Wind
T1	Sunny	hot	Normal	Strong

Program:

```
import pandas as pd
import math
import numpy as np
data = pd.read_csv("C:\\Users\\admin\\Downloads\\3-dataset.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']

        rain
                wind
                        strong ->  ['no']

                        weak ->  ['yes']

        sunny
                humidity
                        high ->  ['no']

                        normal ->  ['yes']

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

Result:

Thus the ID3 algorithm using python program is implemented, executed and output was verified successfully.

EXPNO:3	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
DATE:	

Aim:

To write a python program to build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Algorithm:

BACKPROPAGATION (training_example, η, nin, nout, nhidden)

Each training example is a pair of the form $(\vec{x}\rightarrow,t\rightarrow)$, where $(x\rightarrow)$ is the vector of network input values, $(t\rightarrow)$ and is the vector of target network output values. η is the learning rate (e.g., .05). ni, is the number of network inputs, nhidden the number of units in the hidden layer, and nout the number of output units. The input from unit i into unit j is denoted xji, and the weight from unit i to unit j is denoted wji

- Create a feed-forward network with ni inputs, nhidden hidden units, and nout output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
- For each $(\vec{x}\rightarrow,t\rightarrow)$, in training examples, Do

Propagate the input forward through the network:

1.Input the instance $\vec{x}\rightarrow$, to the network and compute the output ou of every unit u in the network.Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

3. For each hidden unit *h,* calculate its error term δ_h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta\delta_j x_{i,j}$$

Training Examples:

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

Normalize the input

Example	Sleep	Study	Expected %in Exams
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:

```
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:
 [[0.86061523]
 [0.85364038]
 [0.86092633]]
----------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:
 [[0.86086377]
 [0.85388198]
 [0.86117251]]
----------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:
 [[0.86110969]
 [0.85412105]
 [0.8614161 ]]
----------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:
 [[0.86135304]
 [0.85435764]
 [0.86165713]]
----------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:
 [[0.86159385]
 [0.85459178]
 [0.86189566]]
----------Epoch- 5 Ends----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.86159385]
 [0.85459178]
 [0.86189566]]
```

Result:

Thus the Backpropagation algorithm using python program is implemented, executed and output was verified successfully.

EXPNO:4	**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:	

Aim:

To Write a python 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.

Procedure:

Bayes' Theorem is stated as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Where,
P(h|D) is the probability of hypothesis h given the data D. This is called the **posterior probability**.
P(D|h) is the probability of data d given that the hypothesis h was true.
P(h) is the probability of hypothesis h being true. This is called the **prior probability of h.P(D)** is the probability of the data. This is called the **prior probability of D**

After calculating the posterior probability for a number of different hypotheses h, and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any suchmaximally probable hypothesis is called a ***maximum a posteriori (MAP) hypothesis***.

Bayes theorem to calculate the posterior probability of each candidate hypothesis is ***hMAP*** is a MAP hypothesis provided

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)P(h)$$

(Ignoring P(D) since it is a constant)

Gaussian Naive Bayes

A Gaussian Naive Bayes algorithm is a special type of Naïve Bayes algorithm. It's specificallyused when the features have continuous values. It's also assumed that all the features are following a Gaussian distribution i.e., normal distribution

Representation for Gaussian Naive Bayes

We calculate the probabilities for input values for each class using a frequency. With real- valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.

This means that in addition to the probabilities for each class, we must also store the mean andstandard deviations for each input variable for each class.

Gaussian Naive Bayes Model from Data

The probability density function for the normal distribution is defined by two parameters (meanand standard deviation) and calculating the mean and standard deviation values of each input variable (x) for each class value.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$ Mean

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu)^2\right]^{0.5}$$ Standard deviation

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$ Normal distribution

Examples:

- The data set used in this program is the ***Pima Indians Diabetes problem***.
- This data set is comprised of 768 observations of medical details for Pima Indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.
- The attributes are Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome
- Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)

Sample Examples:

Examples	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1

Program:

```
import csv
import random
import math

class NaiveBayes:
    def __init__(self):
        self.classes = {}
        self.features = []
        self.priors = {}

    def train(self, filename):
        with open(filename, 'r') as file:
            reader = csv.reader(file)
            data = list(reader)

        self.features = data[0][:-1]
        data = data[1:]
        random.shuffle(data)

        for row in data:
            label = row[-1]
            if label not in self.classes:
                self.classes[label] = []
                self.priors[label] = 0
            self.classes[label].append(row[:-1])
            self.priors[label] += 1

        for label in self.classes:
            for i in range(len(self.classes[label][0])):
                values = set([row[i] for row in self.classes[label]])
                self.classes[label][i] = values
```

```
for label in self.priors:
        self.priors[label] /= len(data)
 def classify(self, features):
    max_prob = -1
    max_label = None
    for label in self.classes:
        prob = self.priors[label]
        for i in range(len(features)):
            if features[i] in self.classes[label][i]:
                count = sum([1 for row in self.classes[label] if row[i] == features[i]])
                prob *= count / len(self.classes[label])
            else:
                prob = 0
        if prob > max_prob:
            max_prob = prob
            max_label = label
    return max_label

 def test(self, filename):
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        data = list(reader)

    correct = 0
    total = 0
    for row in data:
        features = row[:-1]
        label = row[-1]
        if self.classify(features) == label:
            correct += 1
        total += 1

    accuracy = correct / total
    return accuracy
```

```
# Example usage
nb = NaiveBayes()
nb.train('C:\\Users\\admin\\Downloads\\train.csv')
accuracy = nb.test('C:\\Users\\admin\\Downloads\\test.csv')
print('Accuracy:', accuracy)
```

Output:

Accuracy: 0.50

.

Result:

Thus the Bayesian classifier algorithm using python program is implemented, executed and output was verified successfully.

EXPNO:5	Implement naïve Bayesian Classifier model to classify a set of documents and measure the accuracy, precision, and recall.
DATE:	

Aim:

To write a python program to implement naïve Bayesian Classifier model to classify a set of documents and measure the accuracy, precision, and recall.

Algorithm:

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

where A and B are events and $P(B) \neq 0$.

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

Sample Dataset:

6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	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 : 68.50393700787401%

Result:

Thus the Bayesian classifier algorithm using python program is implemented, executed and output was verified successfully.

EXPNO:6	Write a program to construct a Bayesian network to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.
DATE:	

Aim:

To Write a python program to construct a Bayesian network to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Procedure:

Theory

A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable.

Bayesian network consists of two major parts: a directed acyclic graph and a set of conditional probability distributions

- The directed acyclic graph is a set of random variables represented by nodes.
- The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

For illustration, consider the following example. Suppose we attempt to turn on our computer,but the computer does not start (observation/evidence). We would like to know which of the possible causes of computer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: electricity failure and computer malfunction.
The corresponding directed acyclic graph is depicted in below figure.

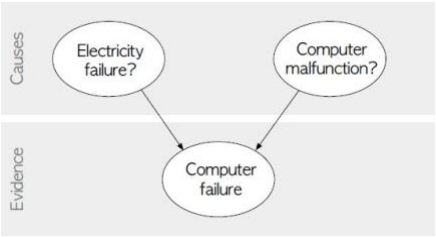


Fig: Directed acyclic graph representing two independent possible causes of a computer failure.

The goal is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. P [Cause | Evidence].

Data Set:
Title: Heart Disease Databases

The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart diseasein the patient. It is integer valued from 0 (no presence) to 4.

Database:	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

Attribute Information:

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
    - Value 1: typical angina
    - Value 2: atypical angina
    - Value 3: non-anginal pain
    - Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevationor depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
13. Heartdisease: It is integer valued from 0 (no presence) to 4.

Some instance from the dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
60	1	4	130	206	0	2	132	1	2.4	2	2	7	4

Program:

```
import bayespy as bp

# Define the Bayesian network
infection = bp.nodes.Bernoulli(0.05)
fever = bp.nodes.Bernoulli(0.9 if infection else 0.1)
cough = bp.nodes.Bernoulli(0.8 if infection else 0.2)
shortness_of_breath = bp.nodes.Bernoulli(0.7 if infection else 0.3)
sore_throat = bp.nodes.Bernoulli(0.6 if infection else 0.4)

# Define the observed data
fever.observe(1)
cough.observe(1)

# Run the inference algorithm
infection.update()

# Print the probabilities
print('Probability of infection:', infection.get_moments()[0])
print('Probability of no infection:', 1 - infection.get_moments()[0])
```

Output:

```
Probability of infection: 0.05000000000000001
Probability of no infection: 0.95
```

.

Result:

Thus the Bayesian Network using python program is implemented, executed and output was verified successfully.

EXPNO:7	**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:	

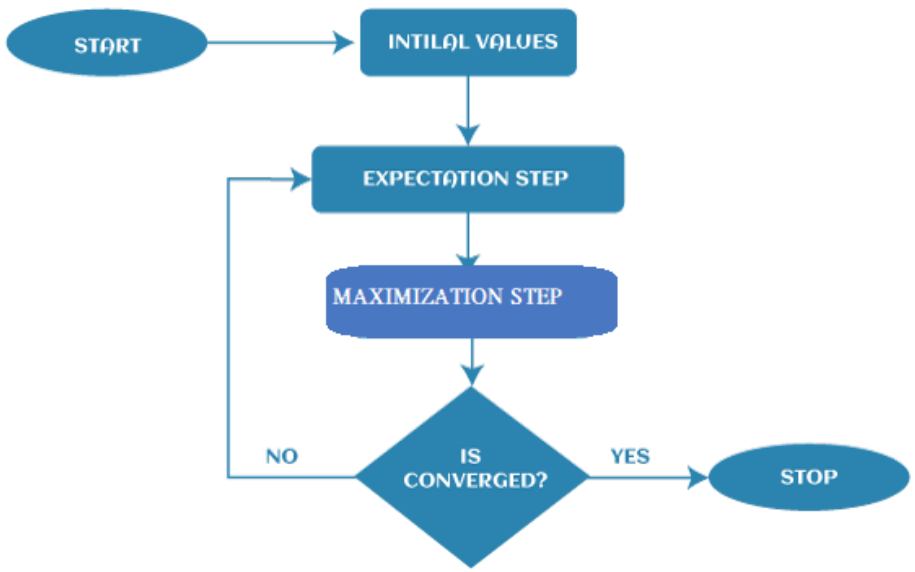
Aim:

To write a python program to apply EM algorithm to cluster a set of data stored in a .CSV file and use the same data set for clustering using the k-Means algorithm and also compare the results of these two algorithms.

Procedure:

EM Algorithm:

The EM algorithm is completed mainly in 4 steps, which include I***nitialization Step, Expectation Step, Maximization Step, and convergence Step***. These steps are explained as follows:



- **1st Step:** The very first step is to initialize the parameter values. Further, the system is provided with incomplete observed data with the assumption that data is obtained from a specific model.
- **2nd Step:** This step is known as Expectation or E-Step, which is used to estimate or guess the values of the missing or incomplete data using the observed data. Further, E-step primarily updates the variables.
- **3rd Step:** This step is known as Maximization or M-step, where we use complete data obtained from the 2nd step to update the parameter values. Further, M-step primarily updates the hypothesis.
- **4th step:** The last step is to check if the values of latent variables are converging or not. If it gets "yes", then stop the process; else, repeat the process from step 2 until the convergence occurs.

K-Means Algorithm:

K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Gaussian Mixture Model:

Demonstration of Gaussian mixture models for classification.

Plots predicted labels on both training and held out test data using a variety of GMM classifiers on the iris dataset.

Compares GMMs with spherical, diagonal, full, and tied covariance matrices in increasing order of performance. Although one would expect full covariance to perform best in general, it is prone to overfitting on small datasets and does not generalize well to held out test data.

On the plots, train data is shown as dots, while test data is shown as crosses. The iris dataset is four-dimensional. Only the first two dimensions are shown here, and thus some points are separated in other dimensions.

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 ("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-------------------------------------------------------------------------")
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 ("-------------------------------------------------------------------------")
```

Output:

The accuracy score of K-Mean: 0.24

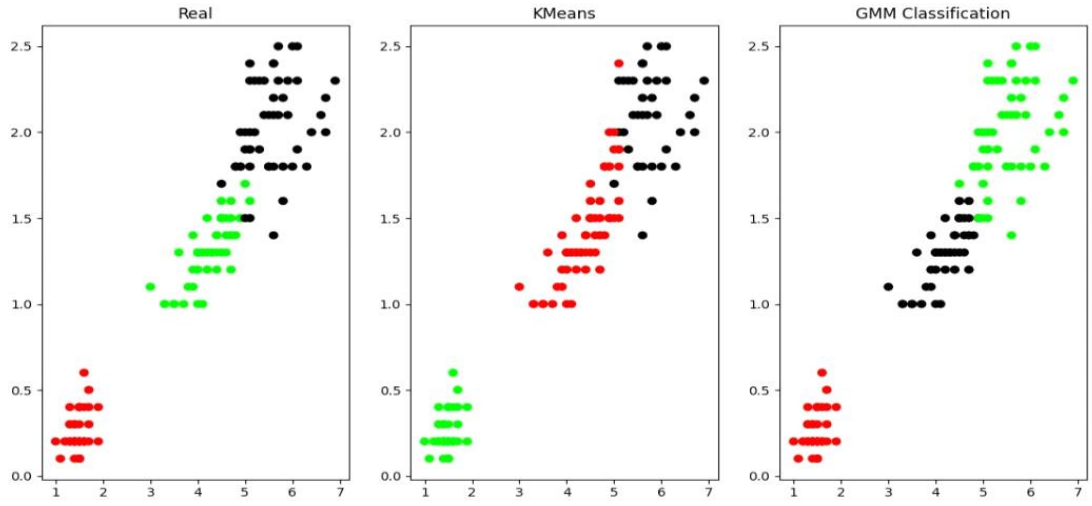
The Confusion matrixof K-Mean:

```
 [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
```

The accuracy score of EM: 0.36666666666666664

The Confusion matrix of EM:

[[50 0 0]
[0 5 45]
[0 50 0]]



Result:

Thus the EM algorithm, K-Means algorithm and compare the results of these two algorithms using python program is implemented, executed and output was verified successfully.

EXPNO:8	Write a program to implement K-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.
DATE:	

Aim:

To write a python program to implement k-Nearest neighbor algorithm to classify the iris data set and also print the both correct and wrong predictions.

Algorithm:

K-Nearest Neighbor Algorithm

Training algorithm:

• For each training example (x, f (x)), add the example to the list training examples Classification algorithm:

• Given a query instance x_q to be classified,

• Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to x_q

• Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

• Where, f(x_i) function to calculate the mean value of the k nearest training examples.

Data Set:

Contains 150 instances (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the Class

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Program:

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
# Load the iris dataset
iris = load_iris()
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3, random_state=42)
# Create a kNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=3)
# Train the classifier on the training set
knn.fit(X_train, y_train)
# Make predictions on the testing set
y_pred = knn.predict(X_test)
# Print the correct and wrong predictions
correct = 0
wrong = 0
for i in range(len(y_test)):
    if y_test[i] == y_pred[i]:
        print("Correct prediction for test instance", i, ": Predicted =", y_pred[i], "Actual =", y_test[i])
        correct += 1
    else:
        print("Wrong prediction for test instance", i, ": Predicted =", y_pred[i], "Actual =", y_test[i])
        wrong += 1

print("\nTotal correct predictions:", correct)
print("Total wrong predictions:", wrong)
```

Output:

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

Confusion Matrix:
 [[4 0 0]
 [0 6 1]
 [0 0 4]]
------------------------------------------------------------------------
```

```
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         4
Iris-versicolor       1.00      0.86      0.92         7
 Iris-virginica       0.80      1.00      0.89         4

       accuracy                           0.93        15
      macro avg       0.93      0.95      0.94        15
   weighted avg       0.95      0.93      0.93        15

------------------------------------------------------------------------
Accuracy of the classifer is 0.93
------------------------------------------------------------------------
```

Result:

Thus the k-Nearest neighbor algorithm using python program is implemented, executed and output was verified successfully.

EXPNO:9	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select an appropriate data set for your experiment and draw graphs.
DATE:	

Aim:

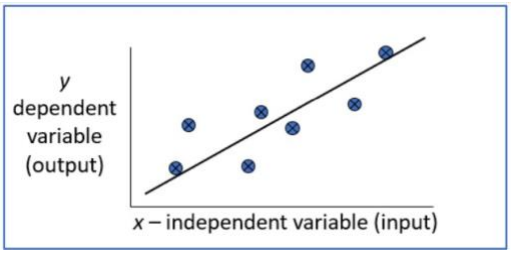
To write a python program to implement the non-parametric Locally Weighted Regression algorithm in order to fit data points and Select an appropriate data set for your experiment and draw graphs.

Algorithm:

Locally Weighted Regression Algorithm

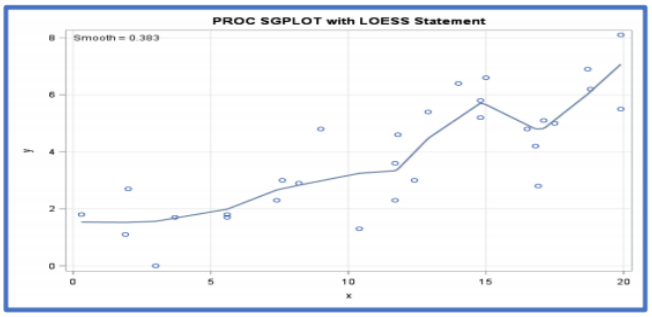
Regression:

- Regression is a technique from statistics that is used to predict values of a desiredtarget quantity when the target quantity is continuous.
- In regression, we seek to identify (or estimate) a continuous variable y associated witha given input vector x.
  - y is called the dependent variable.
  - x is called the independent variable.



Loess/Lowess Regression:

Loess regression is a nonparametric technique that uses local weighted regression to fit asmooth curve through points in a scatter plot.



Lowess Algorithm:

- Locally weighted regression is a very powerful nonparametric model used in statisticallearning.
- Given a dataset X, y, we attempt to find a model parameter β(x) that minimizesresidual sum of weighted squared errors.
- The weights are given by a kernel function (k or w) which can be chosen arbitrarily

Algorithm

1. Read the Given data Sample to X and the curve (linear or non linear) to Y
2. Set the value for Smoothening parameter or Free parameter say τ
3. Set the bias /Point of interest set x0 which is a subset of X
4. Determine the weight matrix using :

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

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

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

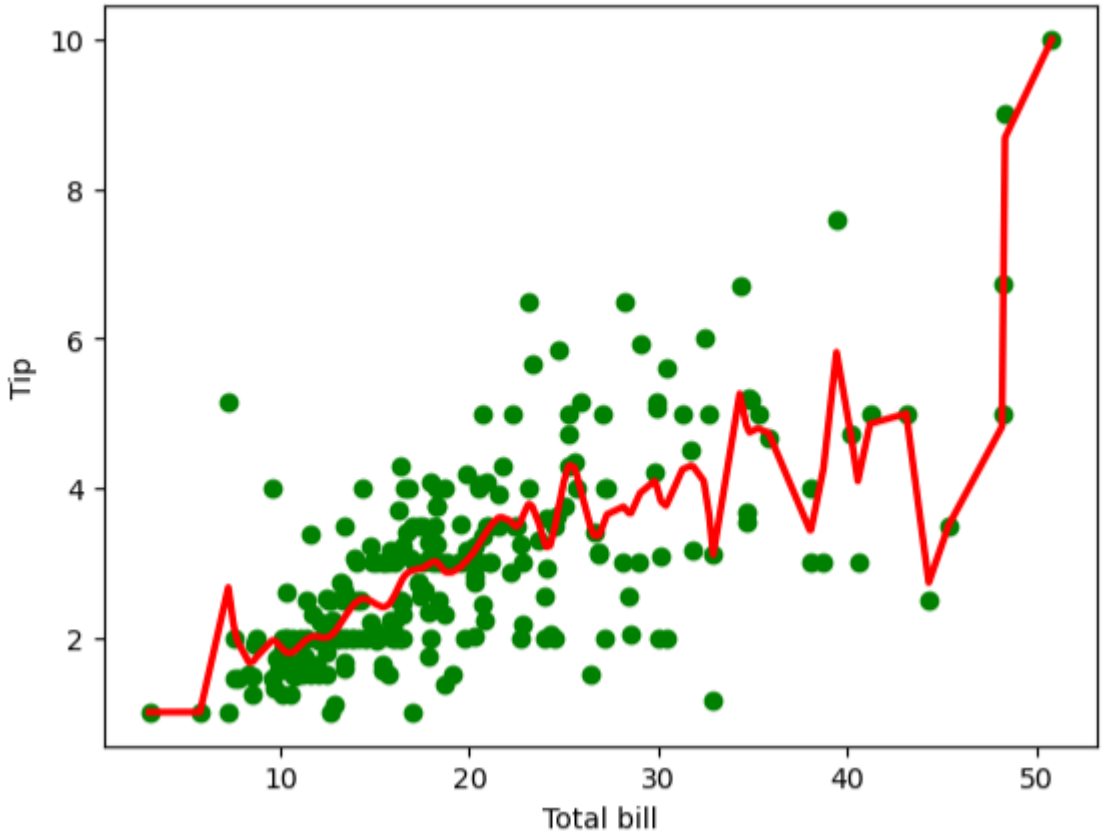
6. Prediction = x0*β:

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=3.5)

plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

Output:



Result:

Thus the Locally Weighted Regression algorithm using python program is implemented, executed and output was verified successfully.