DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE
(AUTONOMOUS)
(Approved by AICTE & Affiliated to Anna University, Chennai)
Re-Accredited by NAAC with 'A' Grade
Accredited by NBA for AERO, BME, CSE, ECE, EEE, IT & MECH.
PERAMBALUR-621212, TAMILNADU, INDIA.
Website: www.dsengg.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

U23CSP42

MACHINE LEARNING LABORATORY

REGULATIONS 2023

YEAR / SEM: II / IV

PREPARED BY **VERIFIED BY**

U23CSP42 MACHINE LEARNING LABORATORY

COURSE OBJECTIVE:

- To understand the python libraries for data science
- To understand the basic Statistical and Probability measures for data science.
- To learn descriptive analytics on the benchmark data sets.
- To apply correlation and regression analytics on standard data sets.
- To present and interpret data using visualization packages in Python.
- Students will develop the ability to build and assess data-based models.

LIST OF EXPERIMENTS

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

2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

3. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

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

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

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

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

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

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

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

COURSE OUTCOME

At the end of the course, the student should be able to:

CO 1: Understand the implementation procedures for the machine learning algorithms.

CO 2: Design Java/Python programs for various Learning algorithms

CO 3: Apply appropriate data sets to the Machine Learning algorithms

CO 4: Apply Machine Learning algorithms to solve real world problems

CO 5: Apply k-Nearest Neighbor algorithm to classify the iris data set.

CO 6: Apply non-parametric Locally Weighted Regression algorithm

LIST OF EXERCISES

S.NO	NAME OF THE EXERCISE	PAGE NO.
1	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.	
2	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.	
3	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.	
4	Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.	
5	Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.	
6	Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.	
7	Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.	
8	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.	
9	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.	
10	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs	

DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE (AUTONOMOUS)

Vision of the Institute

An active and committed centre of advanced learning focused on research and training in the fields of Engineering, Technology and Management to serve the nation better.

Mission of the Institute

M1: To develop eminent scholar with a lifelong follow up of global standards by offering UG, PG and Doctoral Programmes.

M2: To pursue Professional and Career growth by collaborating mutually beneficial partnership with industries and higher institutes of research.

M3: To promote sustained research and training with emphasis on human values and leadership qualities.

M4: To contribute solutions for the need-based issues of our society by proper ways and means as dutiful citizen.

Vision of the Department

To produce globally competent, socially responsible professionals in the field of Computer Science and Engineering.

Mission of the Department

M1: Impart high quality experiential learning to get expertise in modern software tools

M2: Inculcate industry exposure and build inter disciplinary research skills.

M3: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M4: Acquire Innovative skills and promote lifelong learning with a sense of societal and ethical responsibilities

Program Educational Objectives (PEO's)

PEO1: Graduates of the program will develop proficiency in identifying, formulating, and resolving complex computing problems.

PEO2: Graduates of the program will achieve successful careers in the field of computer science and engineering, pursue advanced degrees, or demonstrate entrepreneurial success. **PEO3:** Graduates of the program will cultivate effective communication skills, teamwork abilities, ethical values, and leadership qualities for professional engagement in industry and research organizations.

Program Outcomes (PO'S)

PO1: Engineering knowledge: Apply the basic knowledge of science, mathematics and engineering fundamentals in the field of Computer Science and Engineering to solve complex engineering problems.

PO2: Problem analysis: Ability to use basic principles of mathematics, natural sciences, and engineering sciences to Identify, formulate, review research literature and analyze Computer Science and engineering problems.

PO3: Design/development of solutions: Ability to design solutions for complex Computer Science and engineering problems and basic design system to meet the desired needs within realistic constraints such as manufacturability, durability, reliability, sustainability and economy with appropriate consideration for the public health, safety, cultural, societal, and environmental considerations

PO4: Conduct investigations of complex problems: Ability to execute the experimental activities using research-based knowledge and methods including analyze, interpret the data and results with valid conclusion.

PO5: Modern tool usage: Ability to use state of the art of techniques, skills and modern engineering tools necessary for engineering practice to satisfy the needs of the society with an understanding of the limitations.

PO6: The Engineer and Society: Ability to apply reasoning informed by the contextual knowledge to assess the impact of Computer Science and engineering solutions in legal, health, cultural, safety and societal context and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Ability to understand the professional responsibility and accountability to demonstrate the need for sustainable development globally in Computer Science domain with consideration of environmental effect.

PO8: Ethics: Ability to understand and apply ethical principles and commitment to address the professional ethical responsibilities of an engineer.

PO9: Individual and team work: Ability to function efficiently as an individual or as a group member or leader in a team in multidisciplinary environment.

PO10: Communication: Ability to communicate, comprehend and present effectively with engineering community and the society at large on complex engineering activities by receiving clear instructions for preparing effective reports, design documentation and presentations.

PO11: Project management and finance: Ability to acquire and demonstrate the knowledge of contemporary issues related to finance and managerial skills in one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Ability to recognize and adapt to the emerging field of application in engineering and technology by developing self-confidence for lifelong learning process.

Program Specific Outcome (PSO's)

The graduates of Bachelor of Engineering in Computer Science and Engineering Programme will be able to

PSO1: Analyze, develop and provide solutions to industrial problems in computer domain using programming, data processing and analytical skills.

SO2: Apply software application-oriented skills to innovate solution to meet the ever-changing demands of IT industry.

EX: NO: 1 Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

AIM:

To implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

ALGORITHM

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x For each attribute constraint ai in h If the constraint ai is satisfied by x Then do nothing Else replace ai in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Training Examples:

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 csv
a = []
with open('enjoysport.csv', 'r') as csvfile:
        for row in csv.reader(csvfile):
                a.append(row)
        print(a)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial
hypothesis is : ") hypothesis = ['0']*num_attribute
print(hypothesis)
for i in range(0, len(a)):
        if a[i][num_attribute] == 'yes':
                for j in range(0, num_attribute):
                        if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                                hypothesis[j] = a[i][j]
                        else:
                                hypothesis[j] = '?'
```

```
        print("\n The hypothesis for the training instance {} is : \n" .format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

Data Set:

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:

```
The Given Training Data Set
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']
The total number of training instances are : 4
The initial hypothesis is : ['0', '0', '0', '0', '0', '0']
The hypothesis for the training instance 1 is :
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
The hypothesis for the training instance 2 is :
['sunny', 'warm', '?', 'strong', 'warm', 'same']
The hypothesis for the training instance 3 is :
['sunny', 'warm', '?', 'strong', 'warm', 'same']
The hypothesis for the training instance 4 is : ['sunny', 'warm', '?', 'strong', '?', '?']
The Maximally specific hypothesis for the training instance is ['sunny', 'warm', '?', 'strong', '?', '?']
```

RESULT:

Thus the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples has been implemented successfully.

EX: NO: 2 For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

AIM:

To implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

ALGORITHM

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 Examples:

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('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")
```

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:

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

RESULT:

Thus the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples has been implemented successfully.

EX: NO: 3 Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

AIM:

To write a program to demonstrate the working of the decision tree based ID3 algorithm for building the decision tree and apply this knowledge to classify a new sample.

ALGORITHM

- 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, 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 common value of Target_attribute in Examples
      - Else below this new branch add the subtree ID3(*Examples* v_i, Targe_tattribute, Attributes – {A}))
- End
- Return Root

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	Rain	Cool	Normal	Strong
T2	Sunny	Mild	Normal	Strong

PROGRAM:

```
import math import csv
def load_csv(filename):
        lines=csv.reader(open(filename,"r"));
        dataset = list(lines) headers =
        dataset.pop(0) return dataset,headers
class Node:
        def __init__(self,attribute): self.attribute=attribute
                self.children=[] self.answer=""
def subtables(data,col,delete):
        dic={}
        coldata=[row[col] for row in data]
        attr=list(set(coldata))
        counts=[0]*len(attr) r=len(data)
        c=len(data[0])
        for x in range(len(attr)): for y in range(r):
                        if data[y][col]==attr[x]: counts[x]+=1
        for x in range(len(attr)):
                dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
                pos=0
                for y in range(r):
                        if data[y][col]==attr[x]: if delete:
                                        del data[y][col] dic[attr[x]][pos]=data[y]
                                pos+=1
        return attr,dic
def entropy(S):
        attr=list(set(S))
        if len(attr)==1: return 0
        counts=[0,0]
        for i in range(2):
                counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)
        sums=0
        for cnt in counts:
                sums+=-1*cnt*math.log(cnt,2) return sums
def compute_gain(data,col):
        attr,dic = subtables(data,col,delete=False)
        total_size=len(data)
        entropies=[0]*len(attr)
        ratio=[0]*len(attr)

        total_entropy=entropy([row[-1] for row in data]) for x in range(len(attr)):
```

```
                ratio[x]=len(dic[attr[x]])/(total_size*1.0) entropies[x]=entropy([row[-
                1] for row in
dic[attr[x]]])
                total_entropy-=ratio[x]*entropies[x] return total_entropy
def build_tree(data,features):
        lastcol=[row[-1] for row in data] if(len(set(lastcol)))==1:
                node=Node("") node.answer=lastcol[0]
                return node
        n=len(data[0])-1 gains=[0]*n
        for col in range(n): gains[col]=compute_gain(data,col)
        split=gains.index(max(gains))
        node=Node(features[split])
        fea = features[:split]+features[split+1:] attr,dic=subtables(data,split,delete=True)

        for x in range(len(attr)): child=build_tree(dic[attr[x]],fea)
                node.children.append((attr[x],child))
        return node
def print_tree(node,level):
        if node.answer!="":
                print("          "*level,node.answer) return
        print("          "*level,node.attribute) for value,n in
        node.children:
                print("          "*(level+1),value)
                print_tree(n,level+2)

def classify(node,x_test,features):
        if node.answer!="":
                print(node.answer) return
        pos=features.index(node.attribute) for value, n in
        node.children:
                if x_test[pos]==value: classify(n,x_test,features)
'''Main program'''
dataset,features=load_csv("data3.csv") node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0) testdata,features=load_csv("data3_test.csv") for xtest in
testdata:
        print("The test instance:",xtest)
        print("The label for test instance:",end=" ") classify(node1,xtest,features)
```

OUTPUT

The decision tree for the dataset using ID3 algorithm is

```
Outlook
 rain
     Wind
          strong
               no
          weak
               yes
 overcast
     yes

 sunny
     Humidity
          normal
               yes
          high
               no
```

```
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 to demonstrate the working of the decision tree based ID3 algorithm for building the decision tree and apply this knowledge to classify a new sample has been implemented successfully.

EX: NO: 4 Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

AIM:

To build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

ALGORITHM

- Create a feed – forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
    - For each training examples, Do
        - Propagate the input forward through the network
        - Propagate the errors backward through the network

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=5000             #Setting training iterations
lr=0.1                 #Setting learning rate
inputlayer_neurons = 2                    #number of features in data set
hiddenlayer_neurons = 3                   #number of hidden layers neurons
output_neurons = 1                        #number of neurons at output layer

#weight and bias initialization wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neur
ons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neuron s))
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)

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

# dotproduct of nextlayererror and currentlayerop
    wout += hlayer_act.T.dot(d_output) *lr
      wh += X.T.dot(d_hiddenlayer) *lr

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

OUTPUT:

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

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

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

RESULT:

Thus the program To build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets has been implemented successfully.

EX: NO: 5 Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

AIM:

To 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.

ALGORITHM:

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 such maximally 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 specifically used 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 and standard 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 (mean and 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

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

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

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 contains the all prob of all class of test data
    probabilities = {}
    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 the 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 = 'naivedata.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 768 rows into train=514 and test=254 rows Accuracy of

the classifier is : 71.65354330708661%

RESULT:

Thus the program implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. has been implemented successfully.

EX: NO: 6 Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

AIM:

To write a Built-in Java classes/API program to implement Bayesian Classifier model to perform this task and calculate the accuracy, precision, and recall for your data set.

ALGORITHM:

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms P(wk |vj,), describing the probability that a randomly drawn word from a document in class vj will be the English word wk. It also learns the class prior probabilities P(vj).

1. *collect all words, punctuation, and other tokens that occur in Examples*
   - *Vocabulary* ← ***c*** the set of all distinct words and other tokens occurring in any text document from *Examples*

2. *calculate the required P(vj) and P(wk|vj) probability terms*

   - For each target value *vj* in *V* do
     - *docsj* ← the subset of documents from *Examples* for which the target value is *vj*
     - *P(vj)* ← | *docsj* | / |Examples|
     - *Textj* ← a single document created by concatenating all members of *docsj*
     - *n* ← total number of distinct word positions in *Textj*
     - for each word *wk* in *Vocabulary*
       - *nk* ← number of times word ***wk*** occurs in *Textj*
       - *P(wk|vj)* ← (*nk* + 1) / (n + | *Vocabulary*|)

CLASSIFY_NAIVE_BAYES_TEXT (Doc)

Return the estimated target value for the document Doc. ai denotes the word found in the i^{th} position within Doc.

- *positions* ← all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return *VNB,* where

$$v_{NB} = \operatorname*{argmax}_{v_j \in V} P(v_j) \prod_{i \in positions} P(a_i | v_j)$$

DATA SET:

	Text Documents	Label
1	I love this sandwich	pos
2	This is an amazing place	pos
3	I feel very good about these beers	pos
4	This is my best work	pos
5	What an awesome view	pos
6	I do not like this restaurant	neg
7	I am tired of this stuff	neg
8	I can't deal with this	neg
9	He is my sworn enemy	neg
10	My boss is horrible	neg
11	This is an awesome place	pos
12	I do not like the taste of this juice	neg
13	I love to dance	pos
14	I am sick and tired of this place	neg
15	What a great holiday	pos
16	That is a bad locality to stay	neg
17	We will have good fun tomorrow	pos
18	I went to my enemy's house today	neg

PROGRAM:

```
import pandas as pd msg=pd.read_csv('naivetext.csv',names=['message','label']) print('The dimensions

of the dataset',msg.shape)

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

print(X) print(y)

#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split 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 of count vectoriser is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer count_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_fe ature_names())

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

#printing accuracy, Confusion matrix, Precision and Recall
from sklearn import metrics
print('\n Accuracy of the classifer 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)
0       I love this sandwich
1       This is an amazing place
2       I feel very good about these beers
3       This is my best work
4       What an awesome view
5       I do not like this restaurant
6       I am tired of this stuff
7       I can't deal with this
8       He is my sworn enemy
9       My boss is horrible
10      This is an awesome place
11       I do not like the taste of this juice
12      I love to dance
13      I am sick and tired of this place
14      What a great holiday
15      That is a bad locality to stay
16      We will have good fun tomorrow
17      I went to my enemy's house today
        Name: message, dtype: object
        0   1
        1   1
        2   1
        3   1
        4   1
        5   0
        6   0
        7   0
        8   0
        9   0
        10 1
        11 0
        12 1
        13 0
        14 1
        15 0
        16 1
        17 0
        Name: labelnum, dtype: int64

        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', '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 0.8

Confusion matrix

[[2 1]

[0 2]]

The value of Precision 0.6666666666666666 The value of Recall 1.0

RESULT:

Thus the program to implement Bayesian Classifier model to perform this task and calculate the accuracy, precision, and recall for given data set has been completed successfully.

EX: NO: 7 Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

AIM:

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

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.

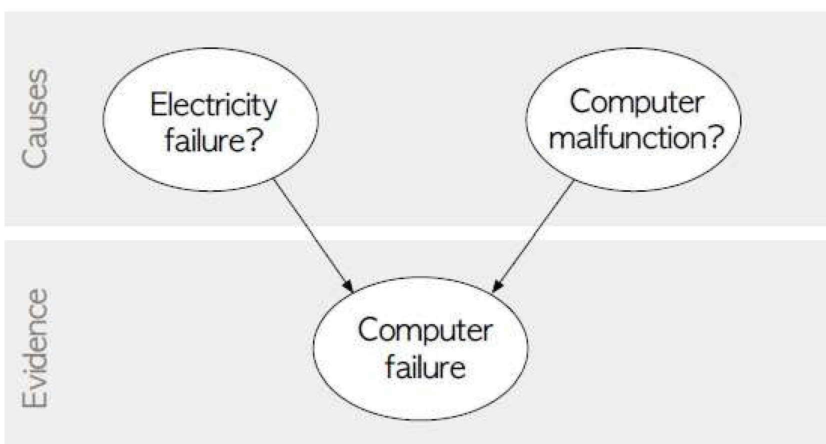


Fig: Directed acyclic graph representing two independent possible causes of a computer failure.

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 disease in 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 elevation or 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 numpy as
np import pandas
as pd import csv
from pgmpy.estimators import MaximumLikelihoodEstimator from
pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination


#read Cleveland Heart Disease data

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

#display the data

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

#display the Attributes names and datatyes


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


#Creat Model- Bayesian Network

model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),(
'exang','heartdisease'),('cp','heartdisease'),('heartdisease', 'restecg'),('heartdisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators


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


#Inferencing with Bayesian Network
```

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

#computing the Probability of HeartDisease given restecg

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

#computing the Probability of HeartDisease given cp

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

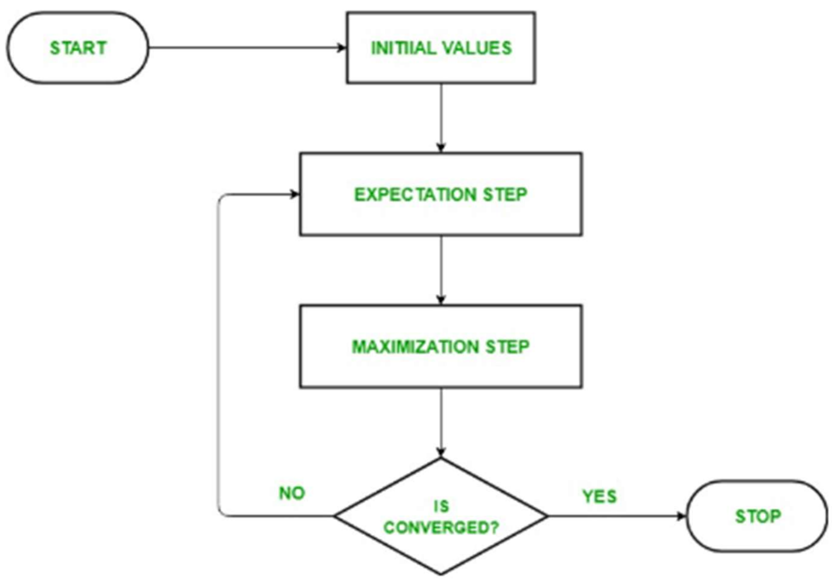
Thus the program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set has been completed successfully.

EX: NO: 8 Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

AIM:

To write a program to implement K-Means clustering algorithm.

ALGORITHM:



EM Algorithm Flowchart

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('Real')
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: 0.36666666666666664

The Confusion matrix of EM:

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

RESULT :

Thus the program to implement K-Means clustering algorithm has been completed successfully.

EX: NO: 9 Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

AIM:

To write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.

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:

Iris Plants Dataset: Dataset 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.model_selection import train_test_split from
sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix from sklearn import
datasets
```

""" Iris Plants Dataset, dataset contains 150 (50 in each of three classes)Number of Attributes: 4 numeric, predictive attributes and the Class

"""

iris=datasets.load_iris()

""" The x variable contains the first four columns of the dataset (i.e. attributes) while y contains the labels.
""""

x =
iris.data y =
iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width') print(x)

print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica') print(y)

""" Splits the dataset into 70% train data and 30% test data. This means that out of total 150 records, the training set will contain

105 records and the test set contains 45 of those records """

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5

classifier = KNeighborsClassifier(n_neighbors=5) classifier.fit(x_train, y_train)

#to make predictions on our test data

y_pred=classifier.predict(x_test)

""" For evaluating an algorithm, confusion matrix, precision, recall and f1 score are the most commonly used metrics.
""""

print('Confusion Matrix') print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics') print(classification_report(y_test,y_pred))

OUTPUT:

```
sepal-length  sepal-width  petal-length  petal-width
[[5.1   3.5   1.4   0.2]
 [4.9   3.    1.4   0.2]
 [4.7   3.2   1.3   0.2]
 [4.6   3.1   1.5   0.2]
 [5.    3.6   1.4   0.2]
 .    .    .    .    .
 .    .    .    .    .

 [6.2   3.4   5.4   2.3]
 [5.9   3.    5.1   1.8]]
```

```
class:  0-Iris-Setosa,  1-  Iris-Versicolour,  2-  Iris-Virginica
[0  0  0  ………0  0  1  1  1  …………1  1  2  2  2  ………… 2  2]
```

```
Confusion Matrix
[[20   0   0]
 [ 0  10   0]
 [ 0   1  14]]
```

Accuracy Metrics

	Precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	0.91	1.00	0.95	10
2	1.00	0.93	0.97	15
avg / total	0.98	0.98	0.98	45

RESULT:

Thus, the program to implement k-Nearest Neighbour algorithm to classify the iris data set has been completed successfully.

EX: NO: 10 Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

AIM:

To write a program to implement the non-parametric Locally Weighted Regression algorithm in order to fit data points

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 :

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

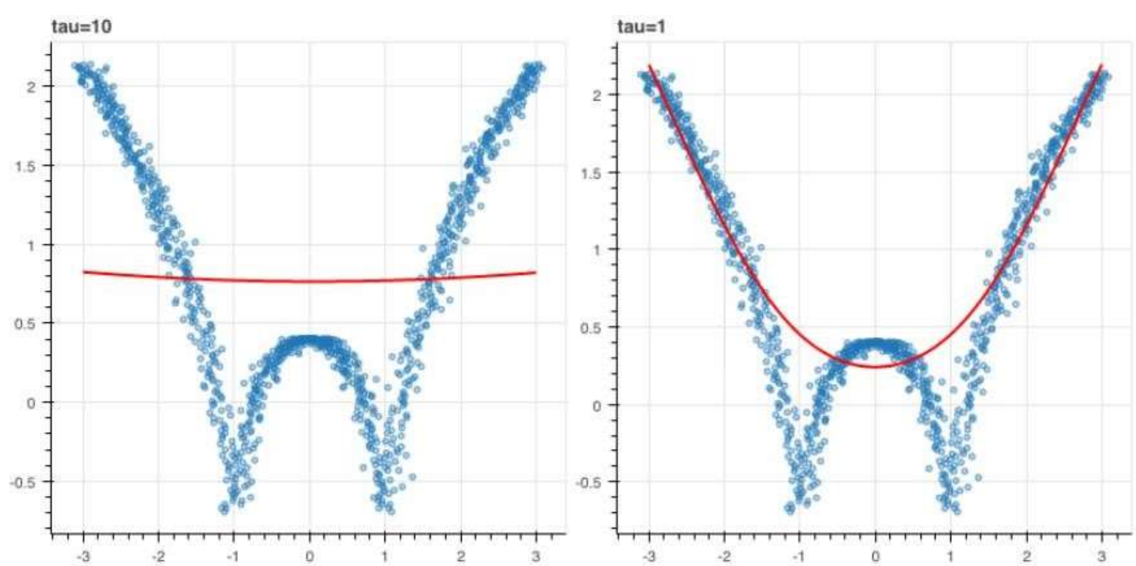
6. Prediction = x0*β:

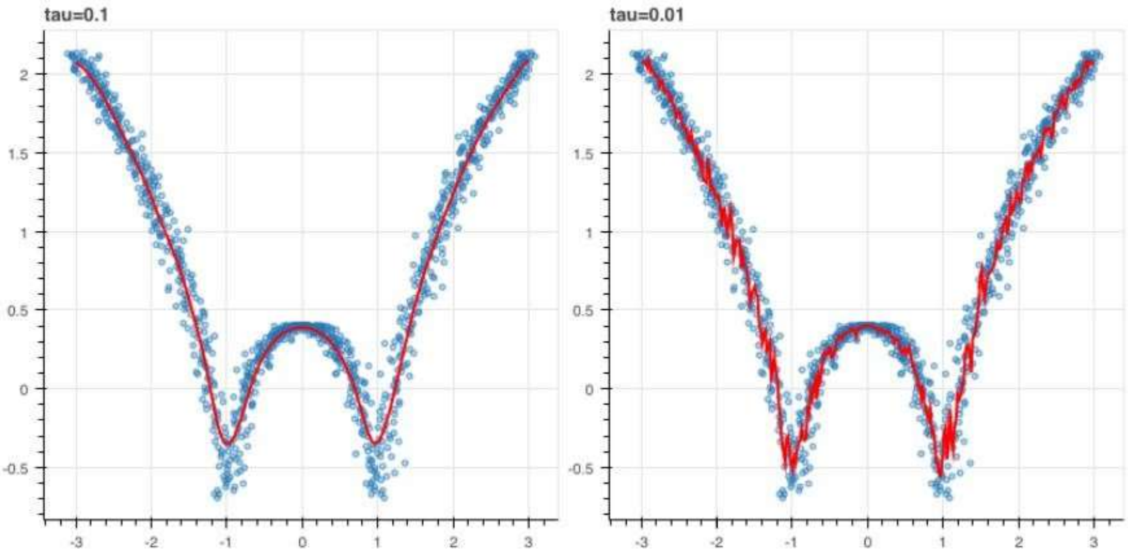
PROGRAM

```
import numpy as np
from bokeh.plotting import figure, show, output_notebook from bokeh.layouts
import gridplot
from bokeh.io import push_notebook

def local_regression(x0, X, Y, tau):# add bias term x0 = np.r_[1, x0]
  # Add one to avoid the loss in
information
  X = np.c_[np.ones(len(X)), X]

  # fit model: normal equations with kernel
  xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W

  beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot
Product

  # predict value
  return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
```

```
def radial_kernel(x0, X, tau):
  return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernal Bias Function

n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10]) Y =
np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y
:\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n) print("Normalised (10
Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10]) def plot_lwr(tau):
  # prediction through regression
  prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
  plot = figure(plot_width=400, plot_height=400) plot.title.text='tau=%g' % tau
  plot.scatter(X, Y, alpha=.3)
  plot.line(domain, prediction, line_width=2, color='red') return plot

show(gridplot([
  [plot_lwr(10.), plot_lwr(1.)],
  [plot_lwr(0.1), plot_lwr(0.01)]]))
```

OUTPUT





RESULT :

Thus the program to implement the non-parametric Locally Weighted Regression algorithm in order to fit data points has been completed successfully.