Experiment 1: Implement and Demonstrate the FIND-S Algorithm

Statement: 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 from CSV file.

Description: By implementing **FIND-S Algorithm,** the most specific hypothesis is found based on given set of training data samples from CSV file.

Algorithm: ***FIND-S Algorithm***

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

Program:

```
import csv
a = [ ]
with open('enjoysport.csv', 'r') as csvfile:
print("The Given Data Set is")
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)
```

Training Sample Data:

Example	Sky	Air Temp	Humidity	Wind	Water	Forecast	Enjoy Sports
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

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', '?', '?']

Experiment2: Implement and Demonstrate the Candidate-Elimination Algorithm

Statement: Implement and Demonstrate the Candidate-Elimination Algorithm for output a description of the set of all hypotheses consistent with the training examples. Read the training data from a .CSV file.

Aim: To implement and demonstrate the Candidate-Elimination algorithm for output a description of the set of all hypotheses consistent with the training examples based on a given set of training data samples from CSV file.

Description: By implementing Candidate-Elimination Algorithm, the set of all hypotheses consistent with the training examples based on a given set of training data samples from CSV file.

Algorithm: Candidate-Elimination *Algorithm*

The Candidate-Elimination 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
```

Program:

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('enjoysport.csv'))
concepts = np.array(data.iloc[:,0:7])
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")
```

Training Sample Data:

Example	Sky	Air Temp	Humidity	Wind	Water	Forecast	Enjoy Sports
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

Output:

```
The Given Data Set is
[['sunny' 'warm' 'high' 'strong' 'warm' 'same' 'yes']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change' 'no']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change' 'yes']]
['yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'high' 'strong' 'warm' 'same' 'yes']
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'high' 'strong' '?' '?' 'yes']
['sunny' 'warm' 'high' 'strong' '?' '?' 'yes']
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' 'high' 'strong' '?' '?' 'yes']
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]
Final Specific_h:
['sunny' 'warm' 'high' 'strong' '?' '?' 'yes']
Final General_h:
[['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?', '?']]
```

Experiment3: Implement and Demonstrate the working of the decision tree based ID3 algorithm.

Statement: 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 implement and demonstrate the working of the decision tree based ID3 algorithm on a given set of training data and test data samples from CSV files to classify a new sample.

Description: By implementing ID3 algorithm, the decision tree will be constructed based on a given set of training data and test data samples from CSV files to classify a new sample.

Algorithm: Decision Tree Based ID3 Algorithm

ID3 determines the information gain for each candidate attribute, and then selects the one with highest information gain as the root node of the tree. The information gain values for all four attributes are calculated using the following formula:

Entropy: Entropy measures the impurity of a collection of examples.

$$\textbf{Entropy(S) = } \Sigma \textbf{ - P(I).log2P(I)}$$

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

$$\textbf{Gain(S, A) = Entropy(S) - } \Sigma \textbf{[P(S/A).Entropy(S/A)]}$$

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. It returns a decision tree that correctly classifies the given Examples.

Create a Root node for the tree

- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples

Otherwise Begin

- A ← the attribute from Attributes that best* classifies Examples
- The decision attribute for Root ←A
- For each possible value, υi, of A,

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

Let Examples υi be the subset of Examples that have value υi for A

If Examples υi 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 υi, Target_attribute, Attributes–{A}))

End

Return Root

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("id3.csv")
node1=build_tree(dataset,features)
print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3_test.csv")
for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end="    ")
    classify(node1,xtest,features)
```

Training Data Sample:

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
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

Testing Data Sample:

Day	**Outlook**	**Temperature**	**Humidity**	**Wind**	**Play Tennis**
D1	Rain	cool	normal	strong	no
D2	Sunny	mild	normal	strong	yes

Output:

The decision tree for the dataset using ID3 algorithm is

Outlook

sunny

Humidity

high

no

normal

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

The test instance: ['rain', 'cool', 'normal', 'strong']

The label for test instance: no

The test instance: ['sunny', 'mild', 'normal', 'strong']

The label for test instance: yes

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

Statement: Write a program to demonstrate the working of the Back-propagation algorithm. Use an appropriate data set for building an Artificial Neural Network.

Aim: To build an Artificial Neural Network using Back-propagation algorithm and on a given set of training data.

Description: By implementing Back-propagation algorithm, the Artificial Neural Network will be constructed based on a given set of training data.

Algorithm:

BACKPROPAGATION Algorithm
BACKPROPAGATION (training_example, n,n_{in},n_{out},n_{hidden})
Each training example is a pair of the form(x,t), where (x) is the vector of network input values, (t) and is the vector of target network output values. n is the learning rate(e.g.,0.05), n_{in} is the number of network inputs, n_{hidden} is the number of units in the hidden layer, and n_{out} is the number of output units.
The input from unit i into j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

Create a feed-forward network with n_{in} 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

Fpr each (x,t), in training examples, Do

Propagate the forward through the network:

1. Input the instance x, to the network and compute the output o_u 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.
$$\delta_h \leftarrow o_{h(1 - o_h)} \Sigma w_{h,k} \delta_k$$
4. Update each network weight W_{ji}

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

Where $\Delta W_{ji} = n \delta_j x_{i,j}$

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)
y = y/100

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

def derivatives_sigmoid(x):
    return x * (1 - x)

epoch=5000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

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))


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)

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

    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

    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)
```

Training Data Sample:

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

Testing Data Sample: (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

Test Result:

Input:

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

Actual Output:

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

Predicted Output:

```
 [[0.88447086]
 [0.86433198]
 [0.88500911]]
```

Experiment 5: Build Naïve Bayesian classifier for a sample training data set stored as a .CSV file

Statement: 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 build naïve Bayesian classifier on a given set of training data.

Description: By implementing Naïve Bayesian classifier, the accuracy will be evaluated on a given set of training data.

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.

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

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

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

Note :- (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

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

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

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)):
                dataset[i] = [float(x) for x in dataset[i]]
        return dataset
def splitdataset(dataset, splitratio):
        trainsize = int(len(dataset) * splitratio);
        trainset = []
        copy = list(dataset);
        while len(trainset) < trainsize:
                index = random.randrange(len(copy));
                trainset.append(copy.pop(index))
        return [trainset, copy]
def separatebyclass(dataset):
        separated = {}
        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):
        summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
        del summaries[-1]
        return summaries
def summarizebyclass(dataset):
```

```
    separated = separatebyclass(dataset);
    summaries = {}
    for classvalue, instances in separated.items():
        summaries[classvalue] = summarize(instances)
    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 = {}
    for classvalue, classsummaries in summaries.items():
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i]
            x = inputvector[i]
            probabilities[classvalue] *= calculateprobability(x, mean, stdev);
    return probabilities
def predict(summaries, inputvector):
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():
        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)))
        summaries = summarizebyclass(trainingset);
        predictions = getpredictions(summaries, testset)
        accuracy = getaccuracy(testset, predictions)
        print('Accuracy of the classifier is : {0}%'.format(accuracy))
 main()
```

Training Data Set:

- 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)

Examples	Pregnancies	Glucose	Blood Pressure	Skin Thickness	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

Output:

Split 768 rows into train=514 and test=254 rows

Accuracy of the classifier is: 74.80314960629921%

Experiment 6: Build Naïve Bayesian classifier for a sample training data set stored as a .CSV file

Statement: 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 build naïve Bayesian classifier on a given set of training data.

Description: By implementing Naïve Bayesian classifier, the accuracy, precision, and recall will be evaluated on a given set of training data.

Algorithm: Naive Bayes algorithms for learning and classifying text

LEARN_NAIVE_BAYES_TEXT (Examples, V)

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

   a) For each target value vj in V do
   b) *docsj* ← the subset of documents from *Examples* for which the target value is *vj*
   c) *P(vj)* ← | *docsj* | / |Examples|
   d) Textj ← a single document created by concatenating all members of docsj
   e) n ← total number of distinct word positions in Textj
   f) for each word *wk* in *Vocabulary*
   g) *nk* ← number of times word ***wk*** occurs in *Textj*
   h) *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 ith position within Doc.

a) positions ← all word positions in Doc that contain tokens found in Vocabulary
b) Return VNB, where

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

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_feature_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 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))
print("\n Accuracy of the Naive Bayes classifer is", metrics.accuracy_score(ytest,predicted))
```

Training 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

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', 'an', 'and', 'awesome', 'bad', 'beers', 'best', 'boss', 'can', 'deal', 'do', 'enemy', 'feel',
'fun', 'good', 'have', 'he', 'horrible', 'is', 'juice', 'like', 'locality', 'my', 'not', 'of', 'place', 'restaurant',
'sick', 'stay', 'stuff', 'sworn', 'taste', 'that', 'the', 'these', 'this', 'tired', 'to', 'tomorrow', 'very', 'view',
'we', 'what', 'will', 'with', 'work']
 Confusion matrix
[[1 0]
 [2 2]]

 The value of Precision 1.0

 The value of Recall 0.5

 Accuracy of the Naive Bayes classifer is 0.6
```

Experiment 7: Bayesian Network Construction for demonstrate the diagnosis of heart patients using standard Heart Disease Data Set that stored as a .CSV file.

Statement: 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 construct Bayesian Network on a given set of training heart disease data.

Description: By constructing Bayesian Network, compute the Probability of Heart Disease given restecg (resting electrocardiographic results) and cp (chest pain type).

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.

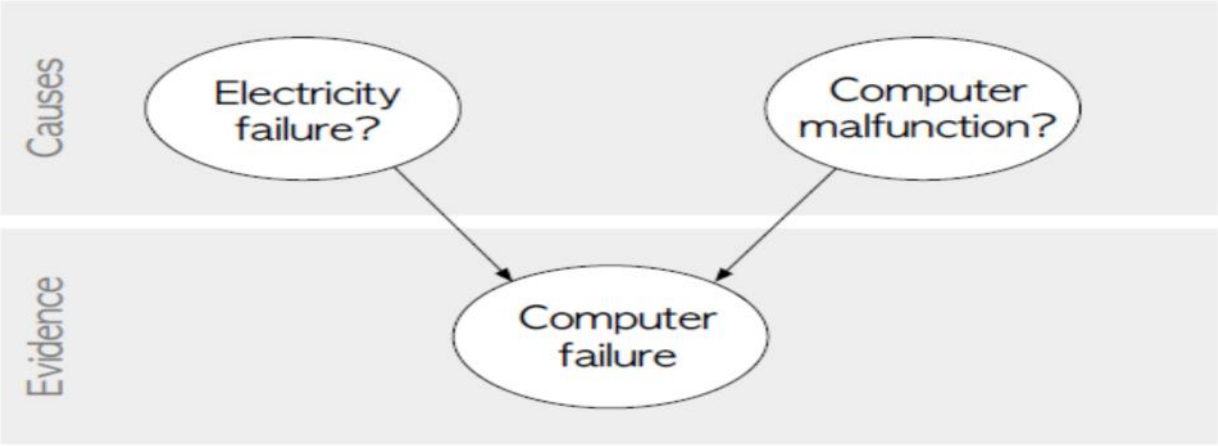


Fig: Directed acyclic graph representing two independent possible causes of a computer failure.

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].

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'],evidence={'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'],evidence={'cp':2})
print(q2)
```

Training 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 "Heart disease" 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	28	0	2	108	1	1.5	2	3	3	2

				6									
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

Output:

```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  ...  oldpeak  slope  cathal  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

Experiment 8: EM algorithm and k-Means Algorithm for clustering a set of data stored in a .CSV file.

Statement: Apply EM (Expectation-Maximization**)** 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 cluster a set of data and comment on quality of clustering using k-Means **Algorithm: Expectation-Maximization**

Description: The EM algorithm tends to get stuck less than K-Means algorithm. The idea is to assign data points partially to different clusters instead of assigning to only one cluster. To do this partial assignment, we model each cluster using a probabilistic distribution. So a data point associates with a cluster with certain probability and it belongs to the cluster with the highest probability in the final assignment. By implementing k-Means Algorithm, the results will be compared and comment on the quality of clustering based on given data set.

Algorithm:

Expectation-Maximization (EM) Algorithm

Step 1: An initial guess is made for the model‟s parameters and a probability distribution is created. This is sometimes called the "E-Step" for the "Expected" distribution.

Step 2: Newly observed data is fed into the model.

Step 3: The probability distribution from the E-step is drawn to include the new data. This is sometimes called the "M-step."

Step 4: Steps 2 through 4 are repeated until stability.

k-Means Algorithm

Step 1: Select *K* points in the data space and mark them as initial centroids

loop

Step 2: Assign each point in the data space to the nearest centroid to form K clusters

Step 3: Measure the distance of each point in the cluster from the centroid

Step 4: Calculate the Sum of Squared Error (SSE) to measure the quality of the clusters

Step 5: Identify the new centroid of each cluster on the basis of distance between points

Step 6: Repeat Steps 2 to 5 to refine until centroids do not change

end loop

Program:

```
import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

# import some data to play with
```

```
iris = datasets.load_iris()
X = pd.DataFrame(iris2.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length',
'Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to
# # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width,
c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
```

```
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')
```

Training Data Set: IRIS Data Set

1	**sepal_length**	**sepal_width**	**petal_length**	**petal_width**	**species**
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	7	3.2	4.7	1.4	versicolor
7	6.4	3.2	4.5	1.5	versicolor
8	6.9	3.1	4.9	1.5	versicolor
9	5.5	2.3	4	1.3	versicolor
10	6.5	2.8	4.6	1.5	versicolor
11	6.3	3.3	6	2.5	virginica
12	5.8	2.7	5.1	1.9	virginica

13	7.6	3	6.6	2.1	virginica
14	4.9	2.5	4.5	1.7	virginica
15	7.3	2.9	6.3	1.8	virginica
16	6.3	3.3	6	2.5	virginica
17	5.8	2.7	5.1	1.9	virginica
18	7.1	3	5.9	2.1	virginica
19	6.3	2.9	5.6	1.8	virginica
20	6.5	3	5.8	2.2	virginica

Experiment 9: k-Nearest Neighbor (k-NN) algorithm for classifying the iris data set.
Statement: Write a program to implement k-Nearest Neighbor (k-NN) 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 classify a set of data and print the prediction information using k-Nearest Neighbor algorithm.
Algorithm: k-NN algorithm
Input: Training data set, test data set (or data points), value of 'k' (i.e. number of nearest neighbors to be considered)
Steps:

Do for all test data points
Calculate the distance (usually Euclidean distance) of the test data point from the different training data points.
Find the closest 'k' training data points, i.e. training data points whose distances are least from the test data point.
If k = 1
Then assign class label of the training data point to the test data point
Else
Whichever class label is predominantly present in the training data points, assign that class label to the test data point
End do

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=datasets.load_iris()

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)

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

Training Data Set: IRIS Data Set (File Name: iris.data)

1	sepal_length	sepal_width	petal_length	petal_width	species
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	7	3.2	4.7	1.4	versicolor
6	6.4	3.2	4.5	1.5	versicolor
7	6.9	3.1	4.9	1.5	versicolor
8	6.3	3.3	6	2.5	virginica
9	5.8	2.7	5.1	1.9	virginica
10	7.6	3	6.6	2.1	virginica
11	4.9	2.5	4.5	1.7	virginica
12	7.3	2.9	6.3	1.8	virginica

Output:

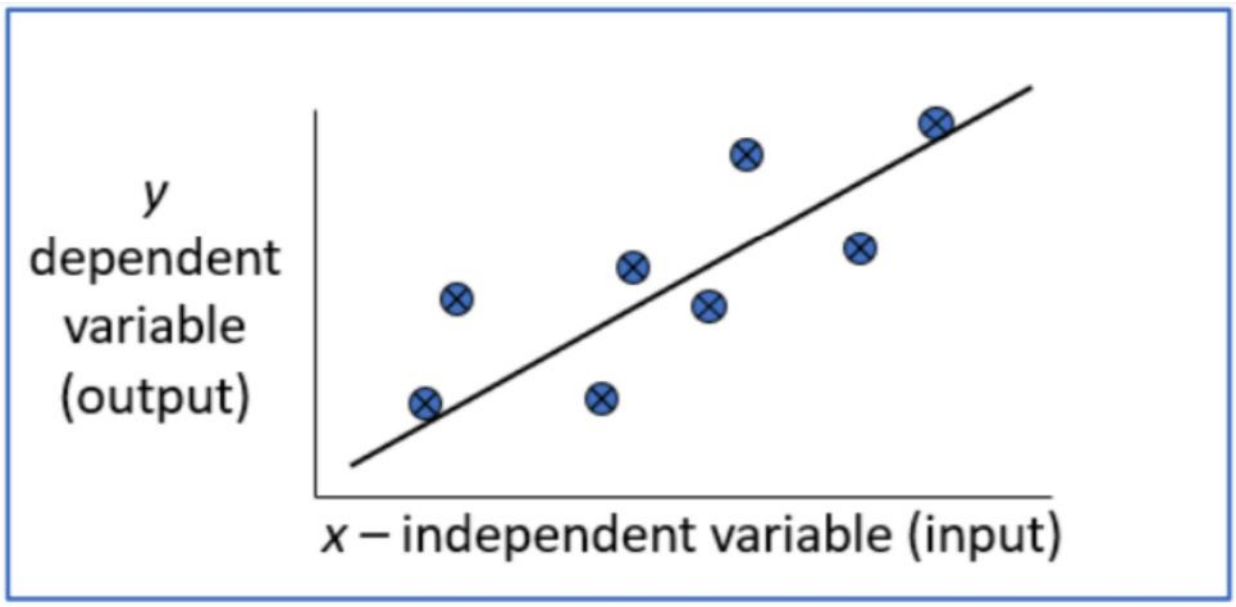
```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.9 3.  5.1 1.8]]
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Confusion Matrix
[[19  0  0]
 [ 0 13  1]
 [ 0  0 12]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.93      0.96        14
           2       0.92      1.00      0.96        12

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.97        45
weighted avg       0.98      0.98      0.98        45
```

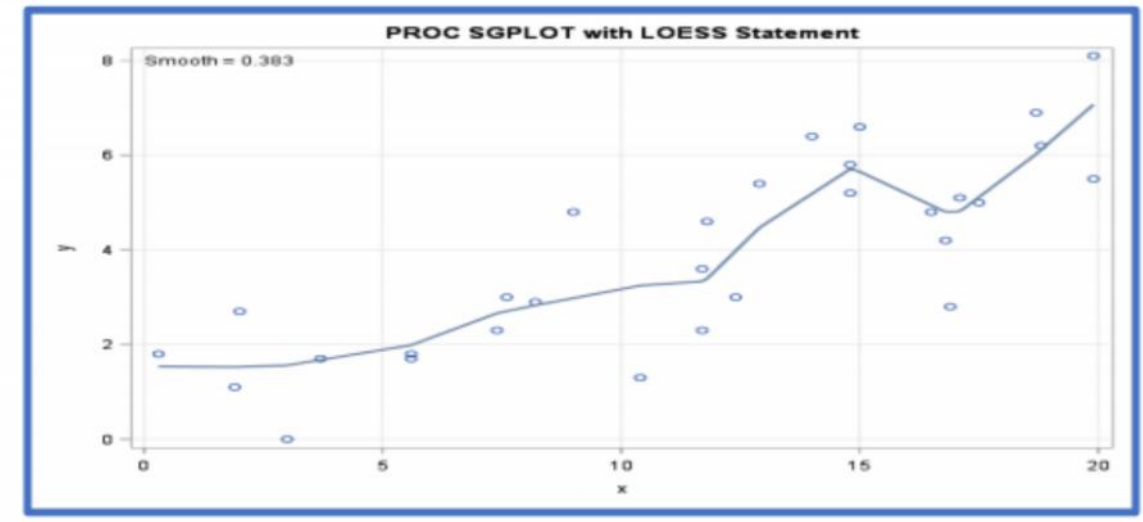
Experiment 10: Implement the non-parametric Locally Weighted Regression algorithm for fitting data points in drawing graphs.
Statement: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.
Aim: To implement the non-parametric Locally Weighted Regression algorithm for fitting data points in drawing graphs for a given set of data.

Algorithm: Locally Weighted Regression Algorithm
Regression:

- Regression is a technique from statistics that is used to predict values of a desired target quantity when the target quantity is continuous.
- In regression, we seek to identify (or estimate) a continuous variable y associated with a 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 a smooth curve through points in a scatter plot.



Lowess Algorithm:

- Locally weighted regression is a very powerful nonparametric model used in statistical learning.
- Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual 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 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:

The Data Set (10 Samples) X :
[-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
-2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
[2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
[-2.89152458 -3.11618584 -3.0400935 -3.0059191 -3.0998411 -2.98624525
-2.93745792 -2.94093534 -2.91410311]
Xo Domain Space(10 Samples) :
[-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
-2.85953177 -2.83946488 -2.81939799]