Geetanjali Institute of Technical Studies

(Approved by AICTE, New Delhi and Affiliated to Rajasthan Technical University Kota (Raj.))

DABOK, UDAIPUR, RAJASTHAN 313022

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

B. Tech - VI SEMESTER

ACADEMIC YEAR – 2024-25

MACHINE LEARNING LAB
6CS4-22

Submitted to: **Submitted By:**

Ms. Upasana Ameta **Name:**

Assistant Professor **Roll No:**

Section:

INDEX

SN	List of Experiments	Date	Signature
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.	09/01/2025	
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.	23/01/2025	
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.	30/01/2025	
4	Build an Artificial Neural Network by implementing the Back-propagation algorithm and test the same using appropriate data sets.	06/02/2025	
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.	13/02/2025	
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.	15/02/2025	
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.	27/02/2025	
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.	06/03/2025	
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.	20/03/2025	
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.	03/04/2025	

EXPERIMENT NO. 1

AIM: 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.

FIND-S Algorithm

Initialize h to the most specific hypothesis in H
For each positive training instance x
For each attribute constraint a_i in h
If the constraint a_i is satisfied by x then do nothing
Else replace a_i in h by the next more general constraint that is satisfied by x
Output hypothesis h

Training Dataset: ML1.CSV

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 pandas as pd

import numpy as np

#to read the data in the csv file
data = pd.read_csv("ML1.csv")
print(data,"n")

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("\n The attributes are: ",d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("\n The target is: ",target)
```

```python
#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

#obtaining the final hypothesis
print("\n The final hypothesis is:",train(d,target))
```

CODE OUTPUT:

```
Sky Air Temp Humidity    Wind Water Forecast Enjoy   Sport
0  Sunny    Warm   Normal  Strong  Warm     Same       Yes
1  Sunny    Warm    High  Strong  Warm     Same       Yes
2  Rainy    Cold    High  Strong  Warm   Change        No
3  Sunny    Warm    High  Strong  Cool   Change       Yes

The attributes are:  [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

The target is:  ['Yes' 'Yes' 'No' 'Yes']
```

EXPERIMENT NO. 2

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

CANDIDATE-ELIMINATION Learning Algorithm

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

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

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

CANDIDATE- ELIMINTION algorithm using version spaces

Training Dataset: ML2.CSV

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('E:/Admin/Desktop/PRACTICALS/ML2.csv'))
concepts = np.array(data.iloc[:,:-1])
target = np.array(data.iloc[:,-1])
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)
        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 ",i+1,"\n ")
        print(specific_h)
        print("general_h ", i+1, "\n ")
        print(general_h)
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

OUTPUT:

```
initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 steps of Candidate Elimination Algorithm 1
Specific_h  1
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
general_h  1
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
 steps of Candidate Elimination Algorithm 2
Specific_h  2
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
general_h  2
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
 steps of Candidate Elimination Algorithm 3
Specific_h  3
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
general_h  3
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' '?' 'Same']
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Specific_h  4
['Sunny' 'Warm' '?' 'Strong' '?' '?']
general_h  4
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']
```

EXPERIMENT NO. 3

AIM: Write a program to demonstrate the working of the decision tree based ID3algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

ID3 Algorithm

ID3 (Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is tobe predicted by the tree. Attributes is a list of other attributes that may be tested by the learneddecisiontree.ReturnsadecisiontreethatcorrectlyclassifiesthegivenExamples.

- Create a Root node for thetree
- 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 inExamples

- Otherwise Begin
  - A ← the attribute from Attributes that best* classifiesExamples
  - The decision attribute for Root ←A
  - For each possible value, v_i, ofA,
    - Add a new tree branch below *Root*, corresponding to the test A =v_i
    - Let *Examples* $_{vi}$, be the subset of Examples that have value $v_i$for*A*
    - If *Examples* $_{vi}$, isempty
      - Then below this new branch add a leaf node with label = most common value of Target_attribute inExamples
      - Else below this new branch add the subtree ID3(*Examples* $_{vi}$, Targe_tattribute, Attributes – {A}))
- End
- ReturnRoot

- The best attribute is the one with highest informationgain

<u>Training Dataset</u>: ML3.CSV

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

Program:

```
import pandas as pd
df = pd.read_csv('E:/Admin/Desktop/PRACTICALS/ML3.csv')
print("\n Input Data Set is:\n", df)
t = df.keys()[-1]
print('Target Attribute is: ', t)
attribute_names = list(df.keys())
attribute_names.remove(t)
print('Predicting Attributes: ', attribute_names)
import math
def entropy(probs):
    return sum( [-prob*math.log(prob, 2) for prob in probs])
def entropy_of_list(ls,value):
    from collections import Counter
    cnt = Counter(x for x in ls)# Counter calculates the propotion of class
    print('Target attribute class count(Yes/No)=',dict(cnt))
    total_instances = len(ls)
    print("Total no of instances/records associated with {0} is: {1}".format(value,total_instances ))
    probs = [x / total_instances for x in cnt.values()]  # x means no of YES/NO
    print("Probability of Class {0} is: {1:.4f}".format(min(cnt),min(probs)))
    print("Probability of Class {0} is: {1:.4f}".format(max(cnt),max(probs)))
    return entropy(probs) # Call Entropy
def information_gain(df, split_attribute, target_attribute,battr):
    print("\n\n-----Information Gain Calculation of ",split_attribute, " --------")
    df_split = df.groupby(split_attribute) # group the data based on attribute values
    glist=[]
    for gname,group in df_split:
```

```
print('Grouped Attribute Values \n',group)
        glist.append(gname)

    glist.reverse()
    nobs = len(df.index) * 1.0
    df_agg1=df_split.agg({target_attribute:lambda x:entropy_of_list(x, glist.pop())})
    df_agg2=df_split.agg({target_attribute :lambda x:len(x)/nobs})

    df_agg1.columns=['Entropy']
    df_agg2.columns=['Proportion']

    # Calculate Information Gain:
    new_entropy = sum( df_agg1['Entropy'] * df_agg2['Proportion'])
    if battr !='S':
        old_entropy = entropy_of_list(df[target_attribute],'S-
'+df.iloc[0][df.columns.get_loc(battr)])
    else:

            old_entropy = entropy_of_list(df[target_attribute],battr)
    return old_entropy - new_entropy
def id3(df, target_attribute, attribute_names, default_class=None,default_attr='S'):

    from collections import Counter
    cnt = Counter(x for x in df[target_attribute])# class of YES /NO

    ## First check: Is this split of the dataset homogeneous?
    if len(cnt) == 1:
        return next(iter(cnt))  # next input data set, or raises StopIteration when EOF is hit.

    ## Second check: Is this split of the dataset empty? if yes, return a default value
    elif df.empty or (not attribute_names):
        return default_class  # Return None for Empty Data Set

    ## Otherwise: This dataset is ready to be devied up!
    else:
        # Get Default Value for next recursive call of this function:
        default_class = max(cnt.keys()) #No of YES and NO Class
        # Compute the Information Gain of the attributes:
        gainz=[]
        for attr in attribute_names:
            ig= information_gain(df, attr, target_attribute,default_attr)
            gainz.append(ig)
            print('Information gain of ',attr,' is : ',ig)

        index_of_max = gainz.index(max(gainz))               # Index of Best Attribute
        best_attr = attribute_names[index_of_max]            # Choose Best Attribute to split on
        print("\nAttribute with the maximum gain is: ", best_attr)
        # Create an empty tree, to be populated in a moment
        tree = {best_attr:{}} # Initiate the tree with best attribute as a node
```

```
        remaining_attribute_names =[i for i in attribute_names if i != best_attr]

        # Split dataset-On each split, recursively call this algorithm.Populate the empty tree
with subtrees, which
        # are the result of the recursive call
            for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,target_attribute,
remaining_attribute_names,default_class,best_attr)
            tree[best_attr][attr_val] = subtree
        return tree
from pprint import pprint
tree = id3(df,t,attribute_names)
print("\nThe Resultant Decision Tree is:")
pprint(tree)
def classify(instance, tree,default=None): # Instance of Play Tennis with Predicted
    attribute = next(iter(tree)) # Outlook/Humidity/Wind
    if instance[attribute] in tree[attribute].keys(): # Value of the attributs in  set of Tree keys
        result = tree[attribute][instance[attribute]]
        if isinstance(result, dict): # this is a tree, delve deeper
            return classify(instance, result)
        else:
            return result # this is a label
    else:
        return default
```

OUTPUT:

```
The Resultant Decision Tree is:
{'Outlook': {'overcast': 'yes',
            'rain': {'wind': {'strong': 'no', 'weak': 'yes'}},
            'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}
```

EXPERIMENT NO. 4

AIM: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate datasets.

BACKPROPAGATION Algorithm

BACKPROPAGATION (*training_example, η, n_{in}, n_{out}, n_{hidden})*

Each training example is a pair of the form ($\vec{x}$,t), where ($\vec{x}$) is the vector ofnetwork input values, (t) and is the vector of target network output values.

η is the learning rate (e.g., .05). n_i, is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit 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_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small randomnumbers
- Until the termination condition is met,Do
    - For each ($\vec{x}$,t), in training examples, Do

  Propagate the input forward through the network:

  1. Input the instance $\vec{x}$ 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

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

4. Update each network weight w_{ji}

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

Where

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

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=7000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
# draws a random range of numbers uniformly of dim x*y
#Forward Propagation
for i in range(epoch):
    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)
#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
# dotproduct of nextlayererror and currentlayerop
    bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *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:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.99999983]
 [0.99999943]
   [0.99999981]]
```

EXPERIMENT NO. 5

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

Bayes' Theorem

P(H/E) = P(E/H) P(H)/P(E)

- H- Hypothesis , E-Event / Evidence
- Bayes' Theorem works on conditional probability
- We have been given that if the event has happened or the event is true, then we have to calculate the probability of Hypothesis on this event.
- Means the chances of happening H when the event E is happened.
- **P(H) -** It is said **priori (A prior probability),** Probability of H before E is happen**.**
- **P(H/E) - Posterior probability,** Probability of E after event E is true.

Training Dataset: Wine Dataset

- The wine dataset contains the results of a chemical analysis of wines grown in a specific area of Italy.
- It contains total 178 samples (data), with 13 chemical analysis (features) recorded for each sample.
- And contains three classes (our target), with no missing values.

Program:

```
import numpy as np
import pandas as pd
from sklearn import datasets
wine = datasets.load_wine()
print ("Features: ", wine.feature_names)
print ("Labels: ", wine.target_names)
X=pd.DataFrame(wine['data'])
print(X.head())
print(wine.data.shape)
y=print (wine.target)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target,
test_size=0.30,random_state=109)
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print(y_pred)
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
from sklearn.metrics import  confusion_matrix
cm=np.array(confusion_matrix(y_test,y_pred))
cm
```

OUTPUT:

```
Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids',
'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines',
'proline']
Labels:  ['class_0' 'class_1' 'class_2']
      0     1     2     3      4     5   ...     7     8     9     10    11      12
0  14.23  1.71  2.43  15.6  127.0  2.80  ...  0.28  2.29  5.64  1.04  3.92  1065.0
1  13.20  1.78  2.14  11.2  100.0  2.65  ...  0.26  1.28  4.38  1.05  3.40  1050.0
2  13.16  2.36  2.67  18.6  101.0  2.80  ...  0.30  2.81  5.68  1.03  3.17  1185.0
3  14.37  1.95  2.50  16.8  113.0  3.85  ...  0.24  2.18  7.80  0.86  3.45  1480.0
4  13.24  2.59  2.87  21.0  118.0  2.80  ...  0.39  1.82  4.32  1.04  2.93   735.0

[5 rows x 13 columns]
(178, 13)
[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 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 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 2 2]
[0 0 1 2 0 1 0 0 1 0 2 2 2 2 0 1 1 0 0 1 2 1 0 2 0 0 1 2 0 1 2 1 1 0 1 1 0
 2 2 0 2 1 0 0 0 2 2 0 1 1 2 0 0 2]
Accuracy: 0.9074074074074074
```

EXPERIMENT NO. 6

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

Naive Bayes algorithms for learning and classifying text

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 $P(w_k|v_j,)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

1. *collect all words, punctuation, and other tokens that occur inExamples*
   - *Vocabulary* ← ***c*** the set of all distinct words and other tokens occurring in any text document from*Examples*
2. *calculate the required $P(v_j)$ and $P(w_k|v_j)$ probabilityterms*
   - For each target value $v_j$in *V*do
     - $docs_j$← the subset of documents from *Examples* for which the target value is*vj*
     - $P(v_j) \leftarrow |docs_j| / |\text{Examples}|$
     - $Text_j$ ← a single document created by concatenating all members of$docs_j$
     - *n* ← total number of distinct word positions in$Text_j$
     - for each word $w_k$in*Vocabulary*
       - n_k← number of times word $\boldsymbol{w_k}$occurs in$Text_j$
       - $P(w_k|v_j) \leftarrow (n_k + 1) / (n + |Vocabulary|)$

Training Dataset: ML6.CSV

	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
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

msg=pd.read_csv('E:/Admin/Desktop/PRACTICALS/ML6.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
#splitting the dataset into train andtestdata
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 asparsematrix
cv =CountVectorizer()
xtrain_dtm = cv.fit_transform(xtrain)
xtest_dtm=cv.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(cv.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=cv.get_feature_names())
print(df)#tabular representation
print(xtrain_dtm) #sparse matrix representation
# Training Naive Bayes (NB) classifier ontrainingdata.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
#printing accuracy, Confusion matrix, PrecisionandRecall
from sklearn import metrics
print('\n The Accuracy of classifier is' , metrics.accuracy_score(ytest, predicted))
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest, predicted))
print('\n The value of Precision' , metrics.precision_score(ytest, predicted))
print('\n The value of Recall' , metrics.recall_score(ytest, predicted))
```

OUTPUT:

```
The dimensions of the dataset (18, 2)
 The total number of Training Data : (13,)
 The total number of Test Data : (5,)
 The words or Tokens in the text documents
['about', 'am', 'amazing', 'an', 'and', 'awesome', 'bad', 'beers', 'can', 'dance', 'deal', 'do', 'enemy', 'feel', 'good', 'great', 'he', 'holiday', 'is', 'juice', 'like', 'locality', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwich', 'sick', 'stay', 'stuff', 'sworn', 'taste', 'that', 'the', 'these', 'this', 'tired', 'to', 'very', 'view', 'what', 'with']
    about  am  amazing  an  and  awesome  ...  tired  to  very  view  what  with
0       1   0        0   0    0        0  ...      0   0     1     0     0     0
1       0   1        0   0    1        0  ...      1   0     0     0     0     0
2       0   0        0   0    0        0  ...      0   0     0     0     0     1
3       0   0        0   0    0        0  ...      0   1     0     0     0     0
4       0   0        0   0    0        0  ...      0   0     0     0     0     0
5       0   0        0   0    0        0  ...      0   0     0     0     0     0
6       0   0        1   1    0        0  ...      0   0     0     0     0     0
7       0   0        0   0    0        0  ...      0   1     0     0     0     0
8       0   1        0   0    0        0  ...      1   0     0     0     0     0
9       0   0        0   0    0        0  ...      0   0     0     0     0     0
10      0   0        0   0    0        0  ...      0   0     0     0     0     0
11      0   0        0   0    0        0  ...      0   0     0     0     1     0
12      0   0        0   1    0        1  ...      0   0     0     1     1     0

[13 rows x 44 columns]

 The Accuracy of classifier is 0.8

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

 The value of Precision 1.0

 The value of Recall 0.6666666666666666
```

EXPERIMENT NO. 7

AIM: 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.

<u>Training Dataset</u>: heartdisease1.CSV

age	Gender	Family	diet	Lifestyle	cholestrol	heartdisease
0	0	1	1	3	0	1
0	1	1	1	3	0	1
1	0	0	0	2	1	1
4	0	1	1	3	2	0
3	1	1	0	0	2	0
2	0	1	1	1	0	1
4	0	1	0	2	0	1
0	0	1	1	3	0	1
3	1	1	0	0	2	0
1	1	0	0	0	2	1
4	1	0	1	2	0	1
4	0	1	1	3	2	0
2	1	0	0	0	0	0
2	0	1	1	1	0	1
3	1	1	0	0	1	0
0	0	1	0	0	2	1
1	1	0	1	2	1	1
3	1	1	1	0	1	0

Program:

```
import pandas as pd
data=pd.read_csv('E:/Admin/Desktop/PRACTICALS/heartdisease1.csv')
heart_disease=pd.DataFrame(data)
print(heart_disease)

from pgmpy.models import BayesianModel
model=BayesianModel([
('age','Lifestyle'),
('Gender','Lifestyle'),
('Family','heartdisease'),
('diet','cholestrol'),
('Lifestyle','diet'),
('cholestrol','heartdisease'),
('diet','cholestrol')
])
```

```
from pgmpy.estimators import MaximumLikelihoodEstimator
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)
from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)

print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }')
print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')
print('For diet Enter { High:0, Medium:1 }')
print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }')
print('For cholesterol Enter { High:0, BorderLine:1, Normal:2 }')

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
    'age':int(input('Enter age :')),
    'Gender':int(input('Enter Gender :')),
    'Family':int(input('Enter Family history :')),
    'diet':int(input('Enter diet :')),
    'Lifestyle':int(input('Enter Lifestyle :')),
    'cholestrol':int(input('Enter cholestrol :'))
    })

print(q['heartdisease'])
```

OUTPUT:

```
    age  Gender  Family  diet  Lifestyle  cholestrol  heartdisease
0     0       0       1     1         3           0             1
1     0       1       1     1         3           0             1
2     1       0       0     0         2           1             1
3     4       0       1     1         3           2             0
4     3       1       1     0         0           2             0
5     2       0       1     1         1           0             1
6     4       0       1     0         2           0             1
7     0       0       1     1         3           0             1
8     3       1       1     0         0           2             0
9     1       1       0     0         0           2             1
10    4       1       0     1         2           0             1
11    4       0       1     1         3           2             0
12    2       1       0     0         0           0             0
13    2       0       1     1         1           0             1
14    3       1       1     0         0           1             0
15    0       0       1     0         0           2             1
16    1       1       0     1         2           1             1
17    3       1       1     1         0           1             0
18    4       0       1     1         3           2             0
For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }
For Gender Enter { Male:0, Female:1 }
For Family History Enter { yes:1, No:0 }
For diet Enter { High:0, Medium:1 }
For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }
For cholesterol Enter { High:0, BorderLine:1, Normal:2 }
Enter age :1
Enter Gender :1
Enter Family history :0
Enter diet :1
Enter Lifestyle :0
Enter cholestrol :1

+----------------+---------------------+
| heartdisease   |   phi(heartdisease) |
+================+=====================+
| heartdisease_0 |              0.0000 |
+----------------+---------------------+
| heartdisease_1 |              1.0000 |
+----------------+---------------------+
```

EXPERIMENT NO. 8

AIM: 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**.**

K-Means Algorithm

1. Load data set
2. Clusters the data into *k* groups where *k* is predefined.
3. Select *k* points at random as cluster centers.
4. Assign objects to their closest cluster center according to the *Euclidean distance* function.
5. Calculate the centroid or mean of all objects in each cluster.
6. Repeat steps 3, 4 and 5 until the same points are assigned to each cluster in consecutive rounds.

EM algorithm

These are the two basic steps of the EM algorithm, namely **E Step or Expectation Step or Estimation Step** and **M Step or Maximization Step**

Estimation step:

- initialize μ_k, $\sum k$ and $\prod_k$ by some random values, or by K means clustering results or by hierarchical clustering results
- Then for those given parameter values,estimate the value of the latent variables(i.e.γ_k)

Maximization Step:

- Update the value of the parameters(i.e. μ_k, $\sum k$ and $\prod_k$) calculated using ML method

1. Load data set
2. Initialize the mean μ_k ,the covariance matrix $\sum k$ and the mixing coefficients $\prod_k$ by some random values. (or other values)
3. Compute the γ_k values for all k.
4. Again Estimate all the parameters using the current γ_k values.
5. Compute log-likelihood function.
6. Put some convergence criterion.

Program:

```
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
iris=load_iris()
x=pd.DataFrame(iris.data, columns=iris.feature_names)
y=pd.DataFrame(iris.target, columns=['target'])
x.head()
colormap=np.array(['red','blue','green'])
plt.title('Actual Clusters')
plt.scatter(x['sepal width (cm)'], x['petal width (cm)'], c=colormap[y.target])
plt.xlabel('sepal width (cm)')
```

```
plt.ylabel('petal width (cm)')
plt.title('KMeans Clusters')
from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components=3).fit(x).predict(x)
plt.scatter(x['sepal width (cm)'], x['petal width (cm)'], c=colormap[gm])
plt.xlabel('sepal width (cm)')
plt.ylabel('petal width (cm)')
plt.title('GaussianMixture Clusters')
from sklearn import metrics as m
print("KMeans Accuracy: ", m.accuracy_score(y, km.labels_))
print("Gausian Mixture: ", m.accuracy_score(y, gm))
```

OUTPUT:

```
1. KMeans Accuracy:  0.8933333333333333
2. Gausian Mixture:  0.3333333333333333
```

EXPERIMENT NO. 9

AIM: 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.

K-Nearest Neighbor Algorithm

Training algorithm:

- For each training example (x, f (x)), add the example to the list trainingexamples

Classification algorithm:

- Given a query instance x_q to beclassified,
  - Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest tox_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.

Training Dataset: IRIS DATASET

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.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
dataset=load_iris()
#print(dataset)
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_stat
e=0)
kn=KNeighborsClassifier(n_neighbors=1)
```

```
kn.fit(X_train,y_train)
for i in range(len(X_test)):
    x=X_test[i]
    x_new=np.array([x])
    prediction=kn.predict(x_new)
    print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",predicti
    on,dataset["target_names"][prediction])
    print(kn.score(X_test,y_test))
```

OUTPUT:

```
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
```

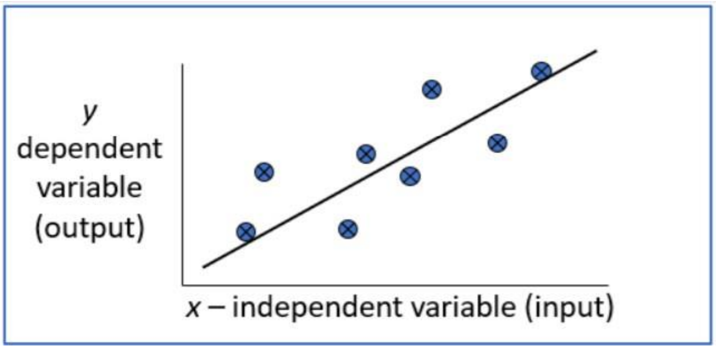
EXPERIMENT NO. 10

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

<u>Locally Weighted Regression Algorithm</u>

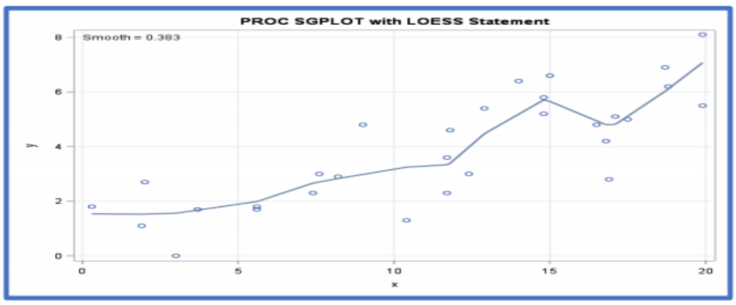
Regression:

- Regression is a technique from statistics that is used to predict values of a desired target quantity when the target quantity iscontinuous.
- In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vectorx.
    - y is called the dependentvariable.
    - x is called the independentvariable.



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 instatistical learning.
- Given a dataset X, y, we attempt to find a model parameter β(x) thatminimizes residual sum of weighted squarederrors.
- The weights are given by a kernel function (k or w) which can be chosenarbitrarily

<u>Algorithm</u>

1. Read the Given data Sample to X and the curve (linear or non linear) toY
2. Set the value for Smoothening parameter or Free parameter sayτ
3. Set the bias /Point of interest set x0 which is a subset ofX

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:

Training Dataset: tips.csv

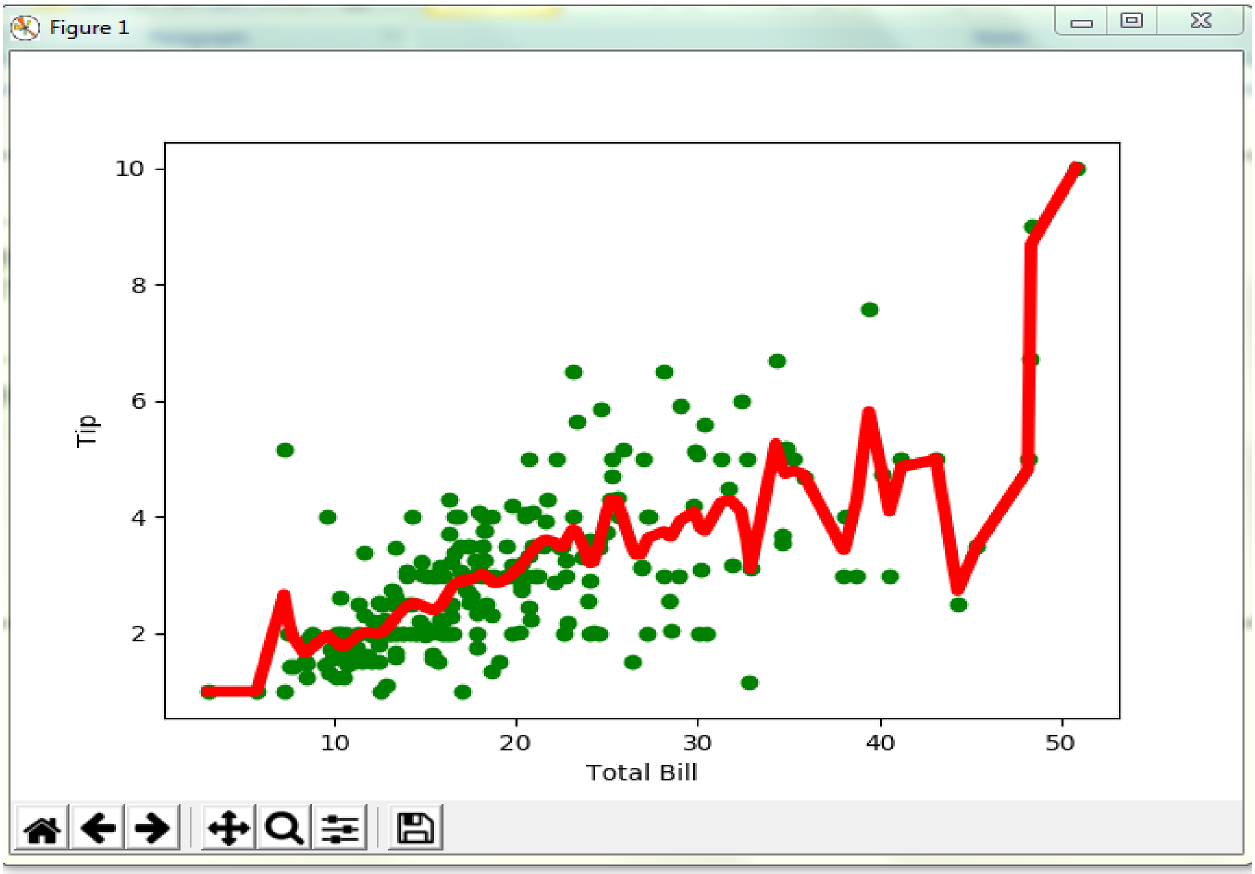
total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2
14.78	3.23	Male	No	Sun	Dinner	2
10.27	1.71	Male	No	Sun	Dinner	2
35.26	5	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2
21.58	3.92	Male	No	Sun	Dinner	2
10.33	1.67	Female	No	Sun	Dinner	3
16.29	3.71	Male	No	Sun	Dinner	3

Program:

```
from numpy import *
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy.linalg
from scipy.stats.stats import pearsonr
def kernel(point,xmat,k):
    m,n= shape(xmat)
    weights=mat(eye((m)))
```

```
    for j in range(m):
        diff = point - X[j]
        weights[j,j]= exp(diff*diff.T/(-2*k**2))
    return weights

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

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

#load data points
data=pd.read_csv('E:/Admin/Desktop/PRACTICALS/tips.csv')
bill=array(data.total_bill)
tip=array(data.tip)

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

#set k here
ypred=localWeightRegression(X,mtip,0.5)
SortIndex=X[:,1].argsort(0)
xsort=X[SortIndex][:,0]
fig=plt.figure()
ax=fig.add_subplot(1,1,1)
ax.scatter(bill,tip,color='green')
ax.plot(xsort[:,1],ypred[SortIndex],color='red',linewidth=5)
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```

OUTPUT:



RUBRICS EVALUATION

Performance Criteria	Scale 1 (0-25%)	Scale 2 (26-50%)	Scale 3 (51-75%)	Scale 4 (76-100%)	Score (Numerical)
Understandability Ability to analyse Problem and Identify solution	Unable to understand the problem.	Able to understand the problem partially and unable to identify the solution	Able to understand the problem completely but unable to identify the solution	Able to understand the problem completely and able to provide alternative solution too.	
Logic Ability to specify Conditions & control flow that are appropriate for the problem domain.	Program logic is incorrect	Program logic is on the right track but has several errors	Program logic is mostly correct, but may contain an occasional boundary error or redundant or contradictory condition.	Program logic is correct, with no known boundary errors, and no redundant or contradictory conditions.	
Debugging Ability to execute /debug	Unable to execute program	Unable to debug several errors.	Able to execute program with several warnings.	Able to execute program completely	
Correctness Ability to code formulae and algorithms that reliably produce correct answers or appropriate results.	Program does not produce correct answers or appropriate results for most inputs.	Program approaches correct answers or appropriate results for most inputs, but can contain miscalculations in some cases.	Program produces correct answers or appropriate results for most inputs.	Program produces correct answers or appropriate results for all inputs tested.	
Completeness Ability to demonstrate and deliver on time.	Unable to explain the code.and the code was overdue.	Unable to explain the code and the code submission was late.	Able to explain code and the program was delivered within the due date.	Able to explain code and the program was delivered on time.	
				TOTAL	