University of Mumbai

PRACTICAL JOURNAL – ELECTIVE II

PSIT3P3a

Machine Learning

SUBMITTED BY

Karkera Prateek Ramesh

SEAT NO 30102

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR QUALIFYING

M.Sc. (I.T.) PART-II (SEMESTER – III) EXAMINATION

2022-2023

Department of Information Technology

3RD FLOOR, DR. SHANKAR DAYAL SHARMA BHAVAN, IDOL

BUILDING, VIDYANAGRI,

SANTACRUZ (E), MUMBAI – 400098.

University of Mumbai

Department of Information Technology

CERTIFICATE

This is to certify that Mr. **Karkera Prateek Ramesh Seat No. 30102** studying in **Master of Science** in **Information Technology Part II Semester III** has satisfactorily completed the Practical of **PSIT3P3a Machine Learning** as prescribed by University of Mumbai, during the academic year **2022-23**.

Signature	Signature	Signature
Guide	External Examiner Examined By	Head of the Department Certified by

College Seal

Date:

INDEX

6	18/11/2022	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.	23	
7	25/11/2022	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	
8	02/12/2022	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.	30	
9	09/12/2022	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.	34	
10	16/12/2022	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs	36	

PRACTICAL 1

TITLE: 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.

CODE:

```
import csv

a = []

with open('enjoysport.csv', 'r') as csvfile:

    for row in csv.reader(csvfile):

        a.append(row)

print(a)

num_attribute = len(a[0]) - 1

print("\n The initial hypothesis is : ")

hypothesis = ['0'] * num_attribute

print(hypothesis)

print("\n The total number of training instances are : ", len(a))

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)
```

OUTPUT:

```
[['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast', 'enjoysport'], ['sunny', 'warm', 'normal', 'strong
 'strong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

 The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

 The total number of training instances are :  5

 The hypothesis for the training instance 1 is :
 ['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 2 is :
 ['sunny', 'warm', 'normal', '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', 'warm', 'same']

 The hypothesis for the training instance 5 is :
 ['sunny', 'warm', '?', 'strong', '?', '?']

 The Maximally specific hypothesis for the training instance is
['sunny', 'warm', '?', 'strong', '?', '?']
```

PRACTICAL 2

TITLE: 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.

CODE:

```
import numpy as np

import pandas as pd



data = pd.read_csv('enjoysport.csv')

concepts = np.array(data.iloc[:, 0:-1])

print(concepts)

target = np.array(data.iloc[:, -1])

print(target)



def learn(concepts, target):

    specific_h = concepts[0].copy()

    print("initialization of specific_h and general_h")

    print(specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]

    print(general_h)



    for i, h in enumerate(concepts):

        print("For Loop Starts")

        if target[i] == "yes":

            print("If instance is Positive ")
```

```
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

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

        print(" steps of Candidate Elimination Algorithm", i + 1)
        print(specific_h)
        print(general_h)
        print("\n")
        print("\n")

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


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

OUTPUT:

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

```
For Loop Starts
If instance is Negative
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



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

PRACTICAL 3

TITLE: 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.

CODE:

```
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("D:/ML_Pracs/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("D:/ML_Pracs/id3_test_1.csv")

for xtest in testdata:

    print("The test instance:", xtest)

    print("The label for test instance:", end=" ")

    classify(node1, xtest, features)
```

OUTPUT:

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
  sunny
   Humidity
    normal
     yes
    high
     no
  rain
   Wind
    strong
     no
    weak
     yes
  overcast
   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
```

PRACTICAL 4

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

CODE:

```
import numpy as np


X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)  # two inputs
[sleep,study]

y = np.array(([92], [86], [89]), dtype=float)  # one output [Expected % in
Exams]

X = X / np.amax(X, axis=0)  # maximum of X array longitudinally

y = y / 100


# Sigmoid Function

def sigmoid(x):

    return 1 / (1 + np.exp(-x))


# Derivative of Sigmoid Function

def derivatives_sigmoid(x):

    return x * (1 - x)


# Variable initialization

epoch = 5000  # Setting training iterations

lr = 0.1  # Setting learning rate

inputlayer_neurons = 2  # number of features in data set

hiddenlayer_neurons = 3  # number of hidden layers neurons
```

```
output_neurons = 1  # number of neurons at output layer

# weight and bias initialization

wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))  # weight of the link from input node to hidden node

bh = np.random.uniform(size=(1, hiddenlayer_neurons))  # bias of the link from input node to hidden node

wout = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))  # weight of the link from hidden node to output node



bout = np.random.uniform(size=(1, output_neurons))  # bias of the link from hidden node to output node

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

for i in range(epoch):

    # Forward Propogation

    hinp1 = np.dot(X, wh)

    hinp = hinp1 + bh

    hlayer_act = sigmoid(hinp)

    outinp1 = np.dot(hlayer_act, wout)

    outinp = outinp1 + bout

    output = sigmoid(outinp)

    # Backpropagation

    EO = y - output

    outgrad = derivatives_sigmoid(output)

    d_output = EO * outgrad

    EH = d_output.dot(wout.T)

    # how much hidden layer weights contributed to error

    hiddengrad = derivatives_sigmoid(hlayer_act)

    d_hiddenlayer = EH * hiddengrad

# dotproduct of nextlayererror and currentlayerop
```

```
wout += hlayer_act.T.dot(d_output) * lr

wh += X.T.dot(d_hiddenlayer) * lr

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n", output)
```

OUTPUT:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.86620261]
 [0.85055724]
 [0.86404415]]
```

PRACTICAL 5

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

CODE:

```
import csv

import random

import math

import pandas as pd


def loadcsv(filename):

    lines = csv.reader(open(filename, "r"));

    dataset = list(lines)

    for i in range(len(dataset)):

    # converting strings into numbers for processing

        dataset[i] = [float(x) for x in dataset[i]]

    return dataset


def splitdataset(dataset, splitratio):

    # 67% training size

    trainsize = int(len(dataset) * splitratio);

    trainset = []

    copy = list(dataset);

    while len(trainset) < trainsize:

    # generate indices for the dataset list randomly to pick ele for
training data
```

```
        index = random.randrange(len(copy));
        trainset.append(copy.pop(index))
    return [trainset, copy]


def separatebyclass(dataset):
    separated = {}  # dictionary of classes 1 and 0
    # creates a dictionary of classes 1 and 0 where the values are
    # the instances belonging to each class
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    print("Separated[0] : ", separated[0])
    print("Separated[1] : ", separated[1])
    return separated


def mean(numbers):
    return sum(numbers) / float(len(numbers))


def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers)
- 1)
    return math.sqrt(variance)


def summarize(dataset):  # creates a dictionary of classes
    summaries = [(mean(attribute), stdev(attribute)) for attribute in
zip(*dataset)];
```

```
    del summaries[-1]  # excluding labels +ve or -ve
    return summaries


def summarizebyclass(dataset):
    separated = separatebyclass(dataset);
    # print(separated)
    summaries = {}
    for classvalue, instances in separated.items():
    # for key,value in dic.items()
    # summaries is a dic of tuples(mean,std) for each class value
        summaries[classvalue] = summarize(instances)  # summarize is used
to cal to mean and std
    return summaries


def calculateprobability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev,
2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent


def calculateclassprobabilities(summaries, inputvector):
    probabilities = {}  # probabilities contains the all prob of all class
of test data
    for classvalue, classsummaries in summaries.items():  # class and
attribute information as mean and sd
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i]  # take mean and sd of every
attribute for class 0 and 1 seperaely
            x = inputvector[i]  # testvector's first attribute
```

```
            probabilities[classvalue] *= calculateprobability(x, mean, stdev);  # use normal dist
    return probabilities


def predict(summaries, inputvector):  # training and test data is passed
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():  # assigns that class which has he highest prob
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel


def getpredictions(summaries, testset):
    predictions = []
    for i in range(len(testset)):
        result = predict(summaries, testset[i])
        predictions.append(result)
    return predictions


def getaccuracy(testset, predictions):
    correct = 0
    for i in range(len(testset)):
        if testset[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(testset))) * 100.0


def main():
```

```
    filename = 'D:/ML_Pracs/naivedata.csv'

    splitratio = 0.67

    dataset = loadcsv(filename);

    dataset1 = pd.read_csv(filename)

    for i in dataset1.columns:

        print("Mean : ",mean(dataset1[i]))

        print("Stdev : ", stdev(dataset1[i]))


    trainingset, testset = splitdataset(dataset, splitratio)

    print('Split {0} rows into train={1} and test={2}
rows'.format(len(dataset), len(trainingset), len(testset)))

    # prepare model

    summaries = summarizebyclass(trainingset);

    print("summaries[0] : ",summaries[0])

    print("summaries[1] : ", summaries[1])

    # test model

    predictions = getpredictions(summaries, testset)  # find the
predictions of test data with the training data

    print('Predictions : ', predictions)

    accuracy = getaccuracy(testset, predictions)

    print('Accuracy of the classifier is : {0}%'.format(accuracy))


main()
```

OUTPUT:

```
Mean :  3.8422425032594525
Stdev :  3.3708765242348493
Mean :  120.85919165580182
Stdev :  31.978468455767796
Mean :  69.10169491525424
Stdev :  19.368154658564634
Mean :  20.517601043024772
Stdev :  15.954059060433842
Mean :  79.90352020860496
Stdev :  115.28310515791267
Mean :  31.990482398956953
Stdev :  7.8890909013660195
Mean :  0.4716740547587999
Stdev :  0.33149735557642684
Mean :  33.21903520208605
Stdev :  11.752295597433893
Mean :  0.34810951760104303
Stdev :  0.47668179499761115
Split 768 rows into train=514 and test=254 rows
Seperated[0] :  [[2.0, 120.0, 76.0, 37.0, 105.0, 39.7, 0.215, 29.0, 0.0], [1.0, 112.0, 80.0, 45.0, 132.0, 34.8, 0.217, 24.0, 0.0], [2.0, 122.0,
Seperated[1] :  [[3.0, 139.0, 54.0, 0.0, 0.0, 25.6, 0.402, 22.0, 1.0], [4.0, 183.0, 0.0, 0.0, 0.0, 28.4, 0.212, 36.0, 1.0], [0.0, 109.0, 88.0,
summaries[0] :  [(3.2222222222222223, 2.949780602762077), (108.66666666666667, 25.412722239321468), (67.70175438596492, 18.211012986117847), (1
summaries[1] :  [(4.575581395348837, 3.7619341810842135), (140.4418604651163, 33.48967010192802), (70.5813953488372, 22.149130441806637), (21.1
Predictions :  [1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.
Accuracy of the classifier is : 68.89763779527559%
```

PRACTICAL 6

TITLE: 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.

CODE:

```
import pandas as pd


msg = pd.read_csv('D:/ML_Pracs/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 the words or Tokens in the text documents
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_out())

df = pd.DataFrame(xtrain_dtm.toarray(),
columns=count_vect.get_feature_names_out())

# Training Naive Bayes (NB) classifier on training data.

from sklearn.naive_bayes import MultinomialNB


clf = MultinomialNB().fit(xtrain_dtm, ytrain)

predicted = clf.predict(xtest_dtm)

# printing accuracy, Confusion matrix, Precision and Recall

from sklearn import metrics


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

```
0     1
1     1
2     1
3     1
4     1
5     0
6     0
7     0
8     0
9     0
10    1
11    0
12    1
13    0
14    1
15    0
16    1
17    0
Name: labelnum, dtype: int64
```

```
The total number of Training Data : (13,)

The total number of Test Data : (5,)

The words or Tokens in the text documents

['about' 'am' 'amazing' 'an' 'and' 'awesome' 'beers' 'best' 'can' 'dance'
 'deal' 'do' 'enemy' 'feel' 'good' 'great' 'he' 'holiday' 'is' 'juice'
 'like' 'love' 'my' 'not' 'of' 'place' 'restaurant' 'sandwich' 'sick'
 'stuff' 'sworn' 'taste' 'the' 'these' 'this' 'tired' 'to' 'very' 'what'
 'with' 'work']

Accuracy of the classifier is 0.4

Confusion matrix
[[0 3]
 [0 2]]

The value of Precision 0.4

The value of Recall 1.0
```

PRACTICAL 7

TITLE: 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.

CODE:

```
import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianNetwork

from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('D:/ML_Pracs/heart.csv')

heartDisease = heartDisease.replace('?',np.nan)

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

print(heartDisease.head())

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

model =
BayesianNetwork([('age','heartdisease'),('sex','heartdisease'),('exang','he
artdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdiseas
e','chol')])

print('\nLearning CPD using Maximum likelihood estimators')

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')

HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
```

```
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'reste
cg':1})

print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')

q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2
})

print(q2)
```

OUTPUT:

```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease
0   63    1   1       145   233  ...      2.3      3   0     6             0
1   67    1   4       160   286  ...      1.5      2   3     3             2
2   67    1   4       120   229  ...      2.6      2   2     7             1
3   37    1   3       130   250  ...      3.5      3   0     3             0
4   41    0   2       130   204  ...      1.4      1   0     3             0

[5 rows x 14 columns]
```

```
 Attributes and datatypes
age               int64
sex               int64
cp                int64
trestbps          int64
chol              int64
fbs               int64
restecg           int64
thalach           int64
exang             int64
oldpeak         float64
slope             int64
ca               object
thal             object
heartdisease      int64
dtype: object
```

```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+----------------+--------------------+
| heartdisease   |   phi(heartdisease) |
+================+====================+
| heartdisease(0) |             0.1012 |
+----------------+--------------------+
| heartdisease(1) |             0.0000 |
+----------------+--------------------+
| heartdisease(2) |             0.2392 |
+----------------+--------------------+
| heartdisease(3) |             0.2015 |
+----------------+--------------------+
| heartdisease(4) |             0.4581 |
+----------------+--------------------+
```

```
 2. Probability of HeartDisease given evidence= cp
+----------------+--------------------+
| heartdisease   |   phi(heartdisease) |
+================+====================+
| heartdisease(0) |             0.3610 |
+----------------+--------------------+
| heartdisease(1) |             0.2159 |
+----------------+--------------------+
| heartdisease(2) |             0.1373 |
+----------------+--------------------+
| heartdisease(3) |             0.1537 |
+----------------+--------------------+
| heartdisease(4) |             0.1321 |
+----------------+--------------------+
```

PRACTICAL 8

TITLE: 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.

CODE:

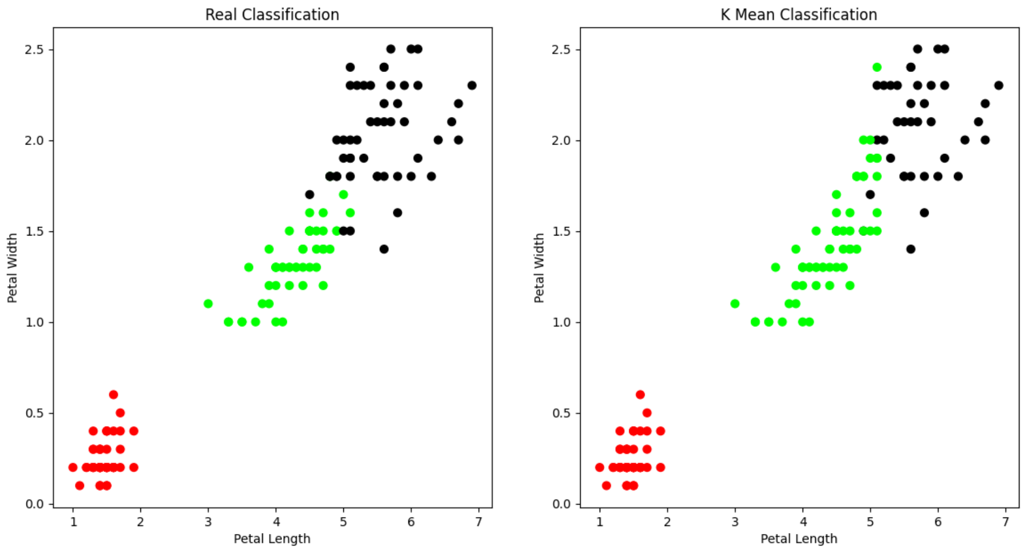
```
import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import sklearn.metrics as sm

import pandas as pd

import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)

X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)

y.columns = ['Targets']

model = KMeans(n_clusters=3)

model.fit(X)

plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications

plt.subplot(1, 2, 1)

plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)

plt.title('Real Classification')

plt.xlabel('Petal Length')
```

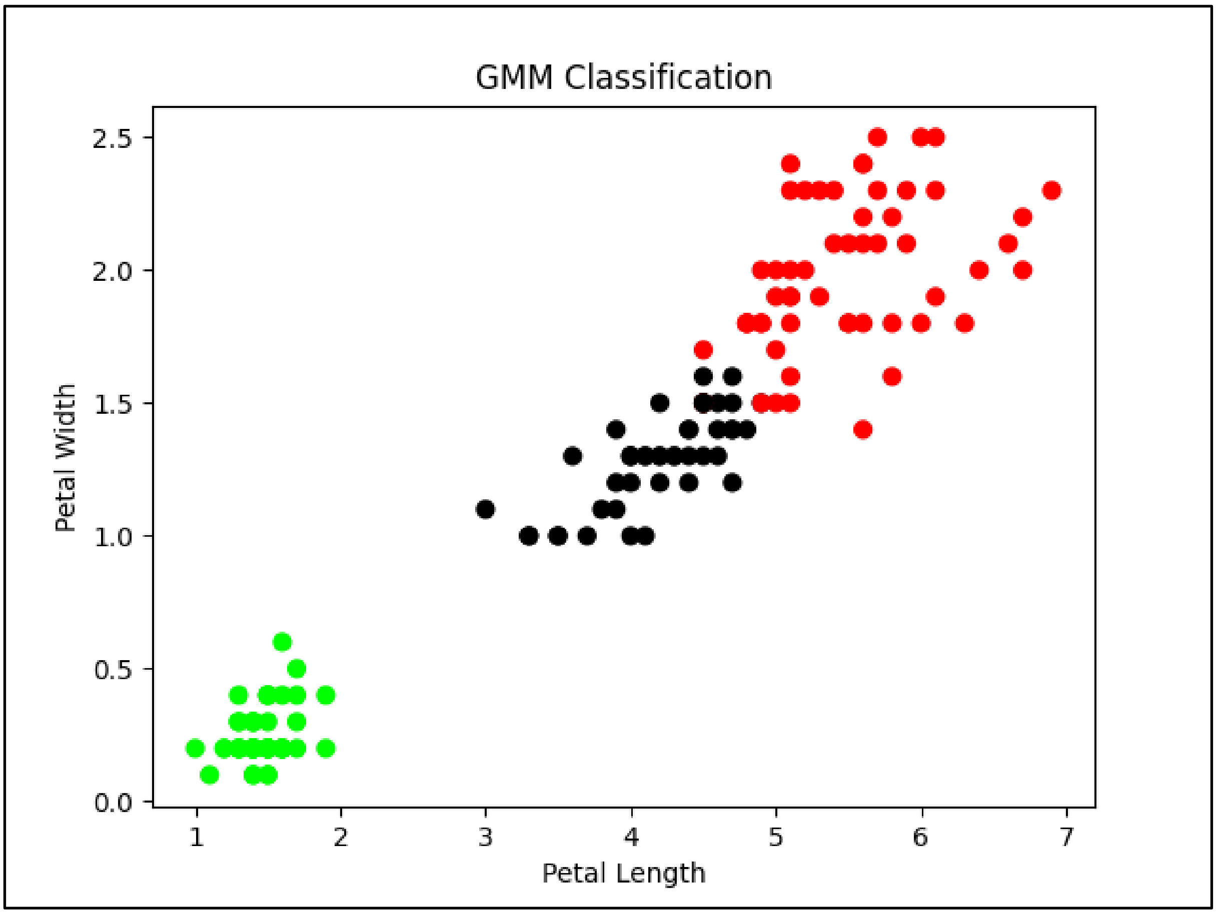
```
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrix of K-Mean: ',sm.confusion_matrix(y,
model.labels_))

from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_gmm = gmm.predict(xs)
#y_cluster_gmm
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
```

```
plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))

print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))

plt.show()
```

OUTPUT:

```
The accuracy score of K-Mean:  0.8933333333333333
The Confusion matrix of K-Mean:  [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
The accuracy score of EM:  0.0
The Confusion matrix of EM:  [[ 0 50  0]
 [ 5  0 45]
 [50  0  0]]
```




GMM Classification
Petal Width
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7
Petal Length

PRACTICAL 9

TITLE: 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.

CODE:

```
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))
```

OUTPUT:

```
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 0 0 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 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 2
 2 2]
Confusion Matrix
[[14  0  0]
 [ 0 14  0]
 [ 0  1 16]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.93      1.00      0.97        14
           2       1.00      0.94      0.97        17

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

PRACTICAL 10

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

CODE:

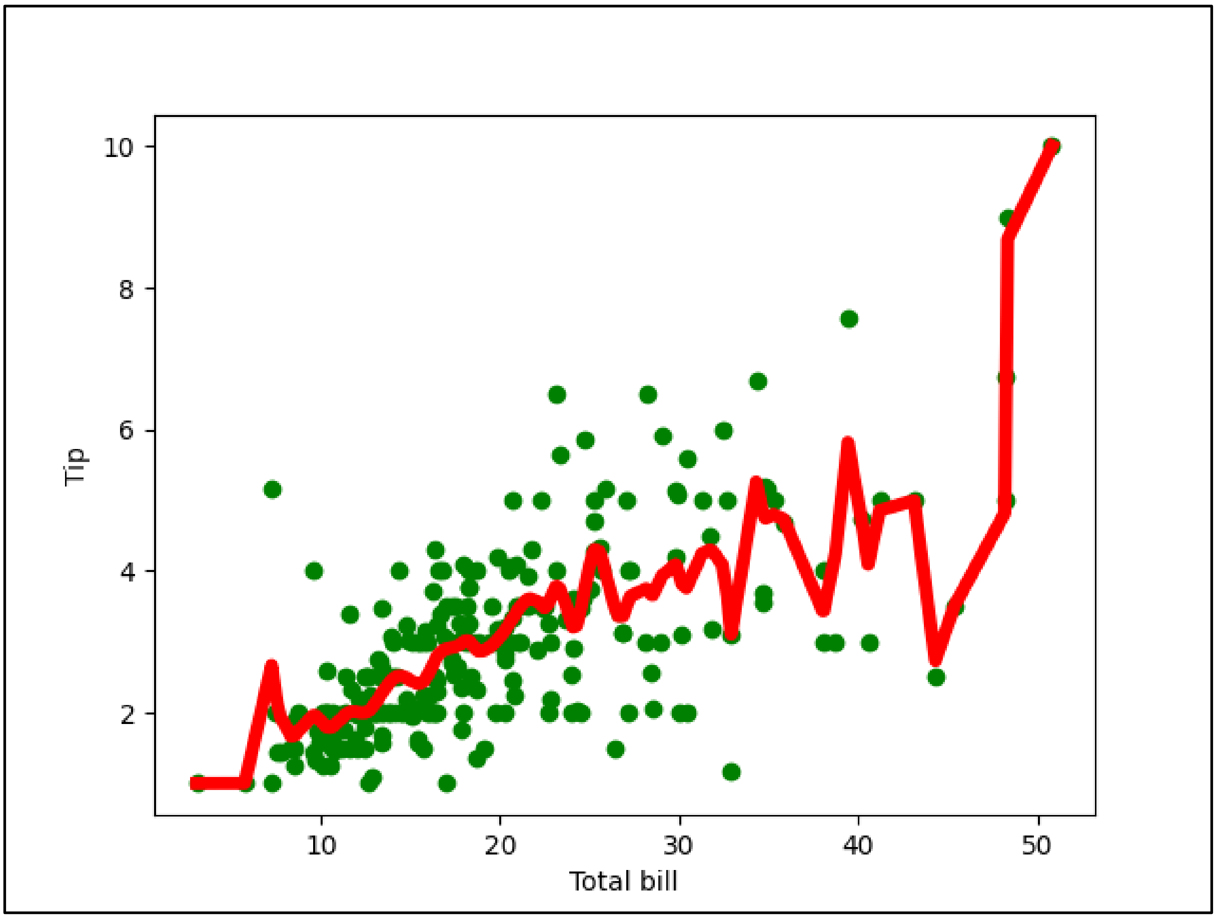
A.

```
import matplotlib.pyplot as plt

import pandas as pd

import numpy as np


def kernel(point, xmat, k):

    m, n = np.shape(xmat)

    weights = np.mat(np.eye((m)))

    for j in range(m):

        diff = point - X[j]

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

    return weights


def localWeight(point, xmat, ymat, k):

    wei = kernel(point, xmat, k)

    W = (X.T * (wei * X)).I * (X.T * (wei * ymat.T))

    return W


def localWeightRegression(xmat, ymat, k):

    m, n = np.shape(xmat)

    ypred = np.zeros(m)
```

```
    for i in range(m):
        ypred[i] = xmat[i] * localWeight(xmat[i], xmat, ymat, k)
    return ypred



# load data points
data = pd.read_csv('D:/ML_Pracs/tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

# preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)

m = np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T, mbill.T))

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

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(bill, tip, color='green')
ax.plot(xsort[:, 1], ypred[SortIndex], color='red', linewidth=5)
plt.xlabel('Total bill')
```

```
plt.ylabel('Tip')

plt.show();
```

A. OUTPUT:

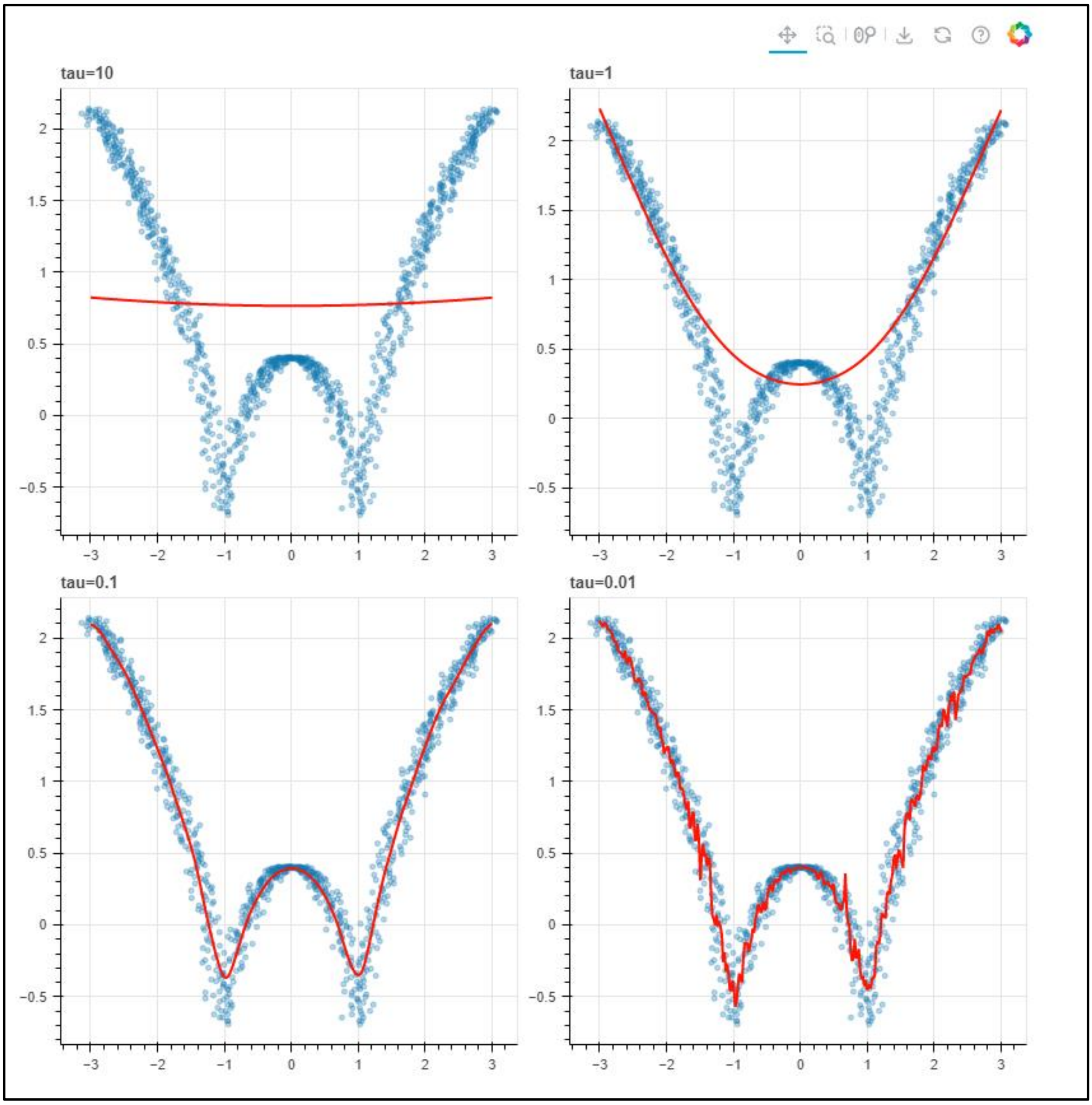


B.

```
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(width=400, 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)]]))
```

B. 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 :
 [-3.02925514 -2.90001609 -2.86223468 -2.78930576 -2.91424246 -2.85378811
 -3.02805408 -3.21185788 -3.03896251]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```