VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING

Submitted by

BASANAGOUDA V B (1BM19CS034)

in partial fulfilment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2022 to July-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled “MACHINE LEARNING” carried out by **BASANAGOUDA V B(1BM19CS034),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Dr. Kayarvizhy N	**Dr. Jyothi S Nayak**
Associate Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

Index Sheet

Machine Learning

Lab1:

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

Program:

```
import pandas as pd
import numpy as np
data = pd.read_csv('lab1.csv')
concepts=np.array(data)[:,:-1]
target=np.array(data)[:,-1]
def search(con,tar):
    for i,val in enumerate(tar):
        if val=="yes":
            specifichyp=con[i].copy()
            break
    for i,val in enumerate(con):
        if tar[i]=="yes":
            for x in range(len(specifichyp)):
                if val[x]!=specifichyp[x]:
                    specifichyp[x]="?"
                else:
                    pass
    return specifichyp
print(search(concepts, target))
```

Output:

```
['sunny' 'warm' '?' 'strong' '?' '?']
```

Lab2:

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.

Program:

```
import numpy as np

import pandas as pd

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

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

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


def candidate_elimination(con,tar):

    s_hyp=con[0].copy()

    g_hyp=[["?" for i in range(len(s_hyp))] for i in range(len(s_hyp))]


    for i,val in enumerate(con):

        if tar[i]=="yes":

            for x in range(len(s_hyp)):

                if val[x]!=s_hyp[x]:

                    s_hyp[x]="?"

                    g_hyp[x][x]="?"

        if tar[i]=="no":

            for x in range(len(s_hyp)):

                if val[x]!=s_hyp[x]:

                    g_hyp[x][x]=s_hyp[x]

                else:

                    g_hyp[x][x]="?"

    indices=[i for i,val in enumerate(g_hyp) if val==["?","?","?","?","?","?"]]

    for i in indices:

        g_hyp.remove(["?","?","?","?","?","?"])
```

```
    return s_hyp,g_hyp

s_final,g_final=candidate_elimination(concepts,target)
print(s_final)
print(g_final)
```

Output:

```
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Lab3:

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.

Program:

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("3-dataset.csv")
features = [feat for feat in data]
features.remove("answer")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
```

```
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain

def ID3(examples, attrs):
    root = Node()
```

```
max_gain = 0
max_feat = ""
for feature in attrs:
    #print ("\n",examples)
    gain = info_gain(examples, feature)
    if gain > max_gain:
        max_gain = gain
        max_feat = feature
root.value = max_feat
#print ("\nMax feature attr",max_feat)
uniq = np.unique(examples[max_feat])
#print ("\n",uniq)
for u in uniq:
    #print ("\n",u)
    subdata = examples[examples[max_feat] == u]
    #print ("\n",subdata)
    if entropy(subdata) == 0.0:
        newNode = Node()
        newNode.isLeaf = True
        newNode.value = u
        newNode.pred = np.unique(subdata["answer"])
        root.children.append(newNode)
    else:
        dummyNode = Node()
        dummyNode.value = u
        new_attrs = attrs.copy()
        new_attrs.remove(max_feat)
        child = ID3(subdata, new_attrs)
        dummyNode.children.append(child)
```

```
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
```

```
outlook
	overcast ->  ['yes']

	rain
		wind
			strong ->  ['no']

			weak ->  ['yes']

	sunny
		humidity
			high ->  ['no']

			normal ->  ['yes']
```

Lab4:

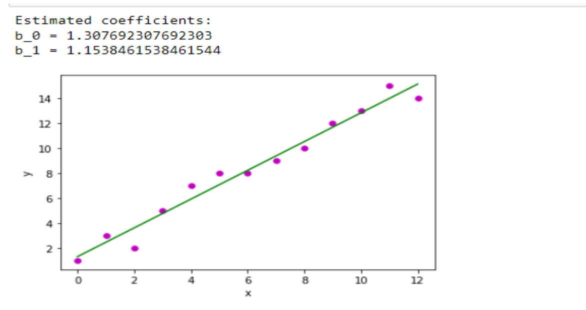
Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Program:

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)
def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
               marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")
```

```
    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12,13,15,14])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
        \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

Output:



Lab5:

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

Program:

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.naive_bayes import MultinomialNB
le=preprocessing.LabelEncoder()
clf = MultinomialNB()
data=pd.read_csv('NB.csv')
features=[feat for feat in data]
targetLabel=features[-1]
features.remove(features[-1])
features
diff_values=[]
for f in features:
    for v in data[f]:
        if v not in diff_values:
            diff_values.append(v)


diff_values
dataArray=np.array(data.iloc[:,0:-1])
dataArray
le.fit(diff_values)
list(le.classes_)
trans=[]
for d in dataArray:
```

```
    trans.append(le.transform(d))
trans
target=data[targetLabel]
target
target=np.array(target)
tar=[]
for t in target:
    if t == "yes":
        tar.append(1)
    else:
        tar.append(0)
tar
clf.fit(trans,tar)
predicting=["sunny","cool","high","strong"]
pre_array=le.transform(predicting)
pre_array=np.reshape(pre_array,(1,4))
pre_array
print(clf.predict(pre_array))
```

Output: [0]

Lab6:

Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

Program:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
# Importing the dataset
```

```
dataset = pd.read_csv('Kmeans_data.csv')
x = dataset.iloc[:, [3, 4]].values
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```

Output:

Lab7:

Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

Program:

```
import pgmpy.models

import pgmpy.inference

import networkx as nx

import pylab as plt

# Create a bayesian network

model = pgmpy.models.BayesianModel([('Burglary', 'Alarm'),

                                   ('Earthquake', 'Alarm'),

                                   ('Alarm', 'JohnCalls'),

                                   ('Alarm', 'MaryCalls')])

# Define conditional probability distributions (CPD)

# Probability of burglary (True, False)

cpd_burglary = pgmpy.factors.discrete.TabularCPD('Burglary', 2, [[0.001], [0.999]])

# Probability of earthquake (True, False)

cpd_earthquake = pgmpy.factors.discrete.TabularCPD('Earthquake', 2, [[0.002], [0.998]])

# Probability of alarm going of (True, False) given a burglary and/or earthquake

cpd_alarm = pgmpy.factors.discrete.TabularCPD('Alarm', 2, [[0.95, 0.94, 0.29, 0.001],

                                                        [0.05, 0.06, 0.71, 0.999]],

                                               evidence=['Burglary', 'Earthquake'],

                                               evidence_card=[2, 2])

# Probability that John calls (True, False) given that the alarm has sounded

cpd_john = pgmpy.factors.discrete.TabularCPD('JohnCalls', 2, [[0.90, 0.05],

                                                        [0.10, 0.95]],

                                               evidence=['Alarm'],

                                               evidence_card=[2])

# Probability that Mary calls (True, False) given that the alarm has sounded

cpd_mary = pgmpy.factors.discrete.TabularCPD('MaryCalls', 2, [[0.70, 0.01],
```

```
                                    [0.30, 0.99]],
                              evidence=['Alarm'],
                              evidence_card=[2])
# Add CPDs to the network structure
model.add_cpds(cpd_burglary, cpd_earthquake, cpd_alarm, cpd_john, cpd_mary)
# Check if the model is valid, throw an exception otherwise
model.check_model()
# Print probability distributions
print('Probability distribution, P(Burglary)')
print(cpd_burglary)
print()
print('Probability distribution, P(Earthquake)')
print(cpd_earthquake)
print()
print('Joint probability distribution, P(Alarm | Burglary, Earthquake)')
print(cpd_alarm)
print()
print('Joint probability distribution, P(JohnCalls | Alarm)')
print(cpd_john)
print()
print('Joint probability distribution, P(MaryCalls | Alarm)')
print(cpd_mary)
print()
# Plot the model
nx.draw(model, with_labels=True)
plt.savefig('C:\\Users\\admin\\Desktop')
plt.close()
# Perform variable elimination for inference
# Variable elimination (VE) is a an exact inference algorithm in bayesian networks
infer = pgmpy.inference.VariableElimination(model)
```

Calculate the probability of a burglary if John and Mary calls (0: True, 1: False)

posterior_probability = infer.query(['Burglary'], evidence={'JohnCalls': 0, 'MaryCalls': 0})

Print posterior probability

print('Posterior probability of Burglary if JohnCalls(True) and MaryCalls(True)')

print(posterior_probability)

print()

Calculate the probability of alarm starting if there is a burglary and an earthquake (0: True, 1: False)

posterior_probability = infer.query(['Alarm'], evidence={'Burglary': 0, 'Earthquake': 0})

Print posterior probability

print('Posterior probability of Alarm sounding if Burglary(True) and Earthquake(True)')

print(posterior_probability)

print()

Output:

```
print()
Probability distribution, P(Burglary)
+-------------+-------+
| Burglary(0) | 0.001 |
+-------------+-------+
| Burglary(1) | 0.999 |
+-------------+-------+

Probability distribution, P(Earthquake)
+---------------+-------+
| Earthquake(0) | 0.002 |
+---------------+-------+
| Earthquake(1) | 0.998 |
+---------------+-------+

Joint probability distribution, P(Alarm | Burglary, Earthquake)
+------------+---------------+---------------+---------------+---------------+
| Burglary   | Burglary(0)   | Burglary(0)   | Burglary(1)   | Burglary(1)   |
+------------+---------------+---------------+---------------+---------------+
| Earthquake | Earthquake(0) | Earthquake(1) | Earthquake(0) | Earthquake(1) |
+------------+---------------+---------------+---------------+---------------+
| Alarm(0)   | 0.95          | 0.94          | 0.29          | 0.001         |
+------------+---------------+---------------+---------------+---------------+
| Alarm(1)   | 0.05          | 0.06          | 0.71          | 0.999         |
+------------+---------------+---------------+---------------+---------------+

Joint probability distribution, P(JohnCalls | Alarm)
+--------------+----------+----------+
| Alarm        | Alarm(0) | Alarm(1) |
+--------------+----------+----------+
| JohnCalls(0) | 0.9      | 0.05     |
+--------------+----------+----------+
| JohnCalls(1) | 0.1      | 0.95     |
+--------------+----------+----------+

Joint probability distribution, P(MaryCalls | Alarm)
+--------------+----------+----------+
| Alarm        | Alarm(0) | Alarm(1) |
+--------------+----------+----------+
| MaryCalls(0) | 0.7      | 0.01     |
+--------------+----------+----------+
| MaryCalls(1) | 0.3      | 0.99     |
+--------------+----------+----------+

Finding Elimination Order: : 0%   0/2 [00:00<?, ?it/s]

Eliminating: Alarm: 100%   2/2 [00:00<00:00, 67.43it/s]
Posterior probability of Burglary if JohnCalls(True) and MaryCalls(True)
+-------------+-----------------+
| Burglary    |   phi(Burglary) |
+=============+=================+
| Burglary(0) |          0.2842 |
+-------------+-----------------+
| Burglary(1) |          0.7158 |
+-------------+-----------------+

Finding Elimination Order: :   0/0 [00:00<?, ?it/s]

  0/0 [00:00<?, ?it/s]
Posterior probability of Alarm sounding if Burglary(True) and Earthquake(True)
+----------+--------------+
| Alarm    |   phi(Alarm) |
+==========+==============+
| Alarm(0) |       0.9500 |
+----------+--------------+
| Alarm(1) |       0.0500 |
+----------+--------------+
```

Lab8:

Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

Program:

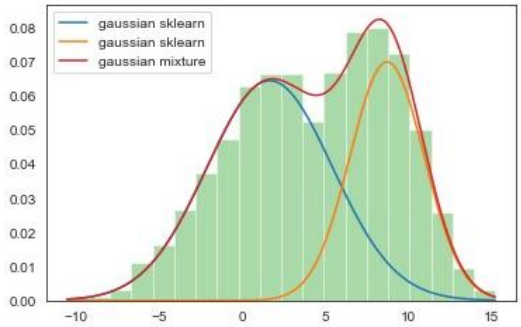
```
# import libraries
# For plotting
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")
%matplotlib inline
#for matrix math
import numpy as np
#for normalization + probability density function computation
from scipy import stats
#for data preprocessing
import pandas as pd
from math import sqrt, log, exp, pi
from random import uniform
print("import done")
random_seed=36788765
np.random.seed(random_seed)

Mean1 = 2.0  # Input parameter, mean of first normal probability distribution
Standard_dev1 = 4.0 #@param {type:"number"}
Mean2 = 9.0 # Input parameter, mean of second normal  probability distribution
Standard_dev2 = 2.0 #@param {type:"number"}

# generate data
y1 = np.random.normal(Mean1, Standard_dev1, 1000)
y2 = np.random.normal(Mean2, Standard_dev2, 500)
```

```
data=np.append(y1,y2)

# For data visiualisation calculate left and right of the graph
Min_graph = min(data)
Max_graph = max(data)
x = np.linspace(Min_graph, Max_graph, 2000) # to plot the data

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1, Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2, Standard_dev2))
sns.distplot(data, bins=20, kde=False)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components = 2, tol=0.000001, max_iter = 100)
gmm.fit(np.expand_dims(data, 1)) # Parameters: array-like, shape (n_samples, n_features), 1 dimension dataset so 1 feature
Gaussian_nr = 1
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1, Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2, Standard_dev2))
for mu, sd, p in zip(gmm.means_.flatten(), np.sqrt(gmm.covariances_.flatten()), gmm.weights_):
    print('Gaussian {:}: μ = {:.2}, σ = {:.2}, weight = {:.2}'.format(Gaussian_nr, mu, sd, p))
    g_s = stats.norm(mu, sd).pdf(x) * p
    plt.plot(x, g_s, label='gaussian sklearn');
    Gaussian_nr += 1
sns.distplot(data, bins=20, kde=False, norm_hist=True)
gmm_sum = np.exp([gmm.score_samples(e.reshape(-1, 1)) for e in x]) #gmm gives log probability, hence the exp() function
plt.plot(x, gmm_sum, label='gaussian mixture');
plt.legend();
```

Output:

```
Input Gaussian 1: μ = 2.0, σ = 4.0
Input Gaussian 2: μ = 9.0, σ = 2.0
Gaussian 1: μ = 1.7, σ = 3.8, weight = 0.61
Gaussian 2: μ = 8.8, σ = 2.2, weight = 0.39
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python310\lib
is a deprecated function and will be removed in a futur
l function with similar flexibility) or `histplot` (an
  warnings.warn(msg, FutureWarning)
```



Lab9:

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

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)
```

Output:

```
Confusion Matrix
[[18  0  0]
 [ 0 17  2]
 [ 0  1  7]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       0.94      0.89      0.92        19
           2       0.78      0.88      0.82         8

    accuracy                           0.93        45
   macro avg       0.91      0.92      0.91        45
weighted avg       0.94      0.93      0.93        45
```

Lab10:

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

Program:

```
from numpy import *
from os import listdir
import matplotlib
```

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr
def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.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 = np1.shape(xmat)
    ypred = np1.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('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip)
# mat is used to convert to n dimesiona to 2 dimensional array form
```

```
m= np1.shape(mbill)[1] # print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
print(X)
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='blue')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

```
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
    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)]]))
```

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr


def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.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 = np1.shape(xmat)
    ypred = np1.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('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,0.3)
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: