VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING

Submitted by

Shivani Gahlot (1BM19CS150)

in partial fulfillment 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 Lab" carried out by **SHIVANI GAHLOT (1BM19CS150),** who is a bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment 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 with respect to **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Saritha A.N. Dr. Jyothi S Nayak Assistant Professor Professor and Head Department of CSE Department of CSE BMSCE, Bengaluru BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.

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

CODE:

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

```
for j in range(0, num_attribute):
if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
hypothesis[j] = a[i][j]
else:
hypothesis[j] = '?'
print("\n The hypothesis for the training instance {} is :\n" .format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instances is :")
print(hypothesis)
```

OUTPUT:

```
The total number of training instances are :  5

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

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 instances is :
['sunny', 'warm', '?', 'strong', '?', '?']
```

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.

CODE:

```
import numpy as np
import pandas as pd

data = pd.read_csv('/kaggle/input/dataset/data.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 :

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.

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)


'''Main program'''
dataset,features=load_csv("/kaggle/input/train/ids_train.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is :\n")
print_tree(node1,0)
```

OUTPUT:

```
The decision tree for the dataset using ID3 algorithm is :

 Outlook
   └─ Rain
     Wind
       └─ Weak
         Yes
       └─ Strong
         No
   └─ Sunny
     Humidity
       └─ Normal
         Yes
       └─ High
         No
   └─ Overcast
     Yes
```

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

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

df = pd.read_csv("/kaggle/input/diabetes/diabetes.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred',
'age'] predicted_class_names = ['diabetes']

X = df[feature_col_names].values
y = df[predicted_class_names].values
print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.40)
```

```
print ('\n The total number of Training Data :',ytrain.shape)
print ('\n The total number of Test Data :',ytest.shape)

clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
print('\n The value of Precision', metrics.precision_score(ytest,predicted))
print('\n The value of Recall', metrics.recall_score(ytest,predicted))
print("Predicted Value for individual Test Data:", predictTestData)
```

OUTPUT:

```
[145 rows x 9 columns]>

 The total number of Training Data : (87, 1)

 The total number of Test Data : (58, 1)

 Confusion matrix
[[31  7]
 [10 10]]

 Accuracy of the classifier is 0.7068965517241379

 The value of Precision 0.5882352941176471

 The value of Recall 0.5
Predicted Value for individual Test Data: [1]
```

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

CODE:

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

# Importing the dataset
dataset = pd.read_csv('/kaggle/input/years-of-experience-and-salary/Years Experience and
```

```
Salary.csv') X = dataset.iloc[:, :-1].values #get a copy of dataset exclude last column
y = dataset.iloc[:, 1].values #get array of dataset in column 1st

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)

# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()


# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

OUTPUT:

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

CODE:

```
#Starting with defining the network structure

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination

#Define a Structure with nodes and edges

cancer_model = BayesianModel([('Pollution', 'Cancer'),
```

```
 ('Smoker', 'Cancer'),
 ('Cancer', 'Xray'),
 ('Cancer', 'Dyspnoea')])
print('Bayesian network nodes:')
print('\t', cancer_model.nodes())
print('Bayesian network edges:')
print('\t', cancer_model.edges())

#Creation of Conditional Probability Table
cpd_poll = TabularCPD(variable='Pollution', variable_card=2,
 values=[[0.9], [0.1]])
cpd_smoke = TabularCPD(variable='Smoker', variable_card=2,
 values=[[0.3], [0.7]])
cpd_cancer = TabularCPD(variable='Cancer', variable_card=2,
 values=[[0.03, 0.05, 0.001, 0.02],
 [0.97, 0.95, 0.999, 0.98]],
 evidence=['Smoker', 'Pollution'],
 evidence_card=[2, 2])
cpd_xray = TabularCPD(variable='Xray', variable_card=2,
 values=[[0.9, 0.2], [0.1, 0.8]],
 evidence=['Cancer'], evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2,
 values=[[0.65, 0.3], [0.35, 0.7]],
 evidence=['Cancer'], evidence_card=[2])
# Associating the parameters with the model structure.
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)
print('Model generated bt adding conditional probability distribution(cpds)')

# Checking if the cpds are valid for the model.
print('Checking for Correctness of model:', end='')
print(cancer_model.check_model())

print('Displaying CPDs')
```

```
print(cancer_model.get_cpds('Pollution'))
print(cancer_model.get_cpds('Smoker'))
print(cancer_model.get_cpds('Cancer'))
print(cancer_model.get_cpds('Xray'))
print(cancer_model.get_cpds('Dyspnoea'))


#Inferencing with Bayesian Network
#Computing the probability of Cancer given smoke

cancer_infer = VariableElimination(cancer_model)
print('\nInferencing with Bayesian Network')

print('\nProbability of Cancer given Smoker')
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1})
print(q)

print('\nProbability of Cancer given Smoker, Pollution')
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1,'Pollution':
1}) print(q)
```

OUTPUT:

```
Inferencing with Bayesian Network

Probability of Cancer given Smoker

Finding Elimination Order: : 0%                                   0/1 [00:00<?, ?it/s]

Eliminating: Pollution: 100%                                   1/1 [00:00<00:00, 30.61it/s]

+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 |
+-----------+---------------+
| Cancer(1) |        0.9971 |
+-----------+---------------+

Probability of Cancer given Smoker, Pollution

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

  0/0 [00:00<?, ?it/s]

+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 |
+-----------+---------------+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```

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

CODE:

```
import pandas as pd
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
df = pd.read_csv('/kaggle/input/income/income.csv')
df.head(10)
scaler = MinMaxScaler()
scaler.fit(df[['Age']])
df[['Age']] = scaler.transform(df[['Age']])
scaler.fit(df[['Income($)']])
df[['Income($)']] = scaler.transform(df[['Income($)']])
df.head(10)
plt.scatter(df['Age'], df['Income($)'])
k_range = range(1, 11)
sse = []
for k in k_range:
 kmc = KMeans(n_clusters=k)
 kmc.fit(df[['Age', 'Income($)']])
 sse.append(kmc.inertia_)
sse
plt.xlabel = 'Number of Clusters'
plt.ylabel = 'Sum of Squared Errors'
plt.plot(k_range, sse)
km = KMeans(n_clusters=3)
km


y_predict = km.fit_predict(df[['Age', 'Income($)']])
y_predict
```

```
df['cluster'] = y_predict
df.head()
df0 = df[df.cluster == 0]
df0
df1 = df[df.cluster == 1]
df1
df2 = df[df.cluster == 2]
df2
km.cluster_centers_

p1 = plt.scatter(df0['Age'], df0['Income($)'], marker='+', color='red')
p2 = plt.scatter(df1['Age'], df1['Income($)'], marker='*', color='blue')
p3 = plt.scatter(df2['Age'], df2['Income($)'], marker='^', color='green')
c = plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color='black')
plt.legend((p1, p2, p3, c),
 ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))
```

OUTPUT:

18	Nick	1.000000	0.162393	2
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	2

[14]:

```
km.cluster_centers_
```

ut[14]:

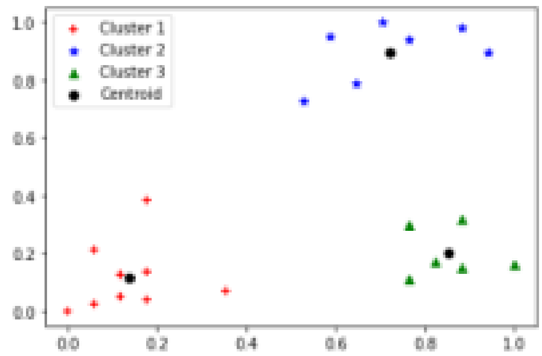
```
array([[0.1372549 , 0.11633428],
       [0.72268908, 0.8974359 ],
       [0.85294118, 0.2022792 ]])
```

[15]:

```
p1 = plt.scatter(df0['Age'], df0['Income($)'], marker='+'
p2 = plt.scatter(df1['Age'], df1['Income($)'], marker='*'
p3 = plt.scatter(df2['Age'], df2['Income($)'], marker='^'
c = plt.scatter(km.cluster_centers_[:,0], km.cluster_cent
plt.legend((p1, p2, p3, c),
           ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centro
```

ut[15]:

```
<matplotlib.legend.Legend at 0x7f518152d950>
```



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

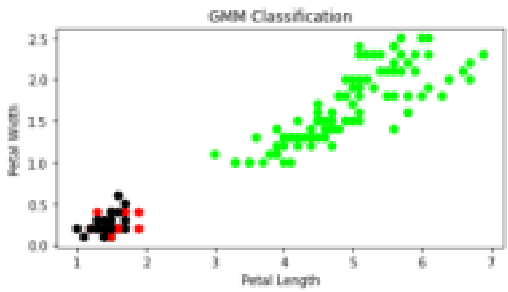
CODE:

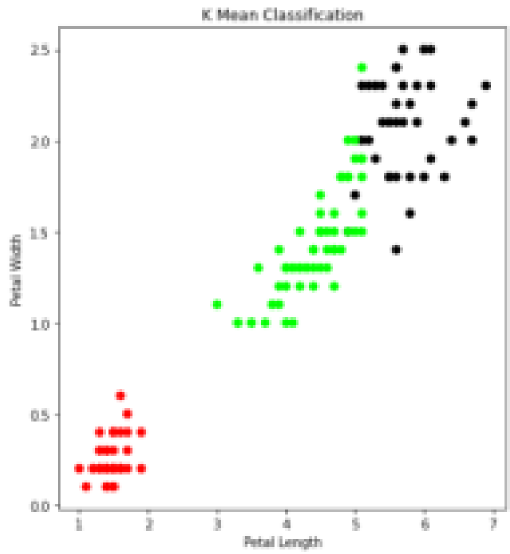
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

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 matrixof 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))
```

OUTPUT:

```
The accuracy score of K-Mean:  0.8933333333333333
The Confusion matrixof K-Mean:  [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
The accuracy score of EM:  0.4
The Confusion matrix of EM:  [[10  0 40]
 [ 0 50  0]
 [ 0 50  0]]
```





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

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:

```
Confusion Matrix
[[19  0  0]
 [ 0 14  2]
 [ 0  0 10]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.88      0.93        16
           2       0.83      1.00      0.91        10

    accuracy                           0.96        45
   macro avg       0.94      0.96      0.95        45
weighted avg       0.96      0.96      0.96        45
```

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.

CODE:

```
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('/kaggle/input/tipsdataset/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)]]))

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))) # eye - identity matrix
 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

def graphPlot(X,ypred):
 sortindex = X[:,1].argsort(0) #argsort - index of the smallest
 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();

# load data points
data = pd.read_csv('/kaggle/input/tipsdataset/tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
```

X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

increase k to get smooth curves

ypred = localWeightRegression(X,mtip,3)

graphPlot(X,ypred)

OUTPUT:

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y:
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.114456
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-3.23963795 -3.01210846 -2.83540045 -3.04102183 -2.96386659 -3.
 -3.0388275  -2.7336852  -3.08914491]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.
 -2.85953177 -2.83946488 -2.81939799]
```