VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING
(20CS6PCMAL)

Submitted by

JOEL NINAN JOHNSON (1BM19CS199)

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**" carried out by **JOEL NINAN JOHNSON (1BM19CS199),** who is 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 in respect of a **Machine Learning- (20CS6PCMAL)** work prescribed for the said degree.

Dr. Asha G. R.
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S. N.
Professor and Head
Department of CSE
BMSCE, Bengaluru

INDEX

Course Outcome

CO4	Ability to conduct practical experiments to solve problems using appropriate machine learning techniques.

LAB PROGRAM 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('enjoysport.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 instance 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 instance is
['sunny', 'warm', '?', 'strong', '?', '?']
```

LAB PROGRAM 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.DataFrame(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 hypothesis and general hypothesis:")
    print("S0: ",specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("G0: ",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] = "?"
        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("\nThe steps of Candidate Elimination Algorithm ", i+1)
        print(f"S{i+1}: " ,specific_h)
        actual_general_h= [i for i in general_h if i != ['?', '?', '?', '?', '?', '?']]
        print(f"G{i+1}: ",actual_general_h)
    actual_general_h= [i for i in general_h if i != ["?" for i in range(len(specific_h))]]
    return specific_h, actual_general_h
```

Output:

```
Initialization of specific hypothesis and general hypothesis:
S0:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
G0:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

The steps of Candidate Elimination Algorithm  1
S1:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
G1:  []

The steps of Candidate Elimination Algorithm  2
S2:  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
G2:  []

The steps of Candidate Elimination Algorithm  3
S3:  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
G3:  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', 'same']]

The steps of Candidate Elimination Algorithm  4
S4:  ['sunny' 'warm' '?' 'strong' '?' '?']
G4:  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?',
'?']]

The final Specific hypothesis is:
['sunny' 'warm' '?' 'strong' '?' '?']
The final General Hypothesis is:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

LAB PROGRAM 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
In [6]:
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)
In [7]:
def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
'''Main program'''
dataset,features=load_csv("id3-dataset.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("id1.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
   rain
     Wind
       weak
         yes
       strong
         no
   overcast
     yes
   sunny
     Humidity
       normal
         yes
       high
         no
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
```

LAB PROGRAM 4:

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 set.

Code:

```
from sklearn.datasets import fetch_20newsgroups
categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
news_train = fetch_20newsgroups(subset='train',categories=categories)
news_test = fetch_20newsgroups(subset='test',categories=categories)
print(news_train.keys())
print(news_train['target_names'])
from sklearn.feature_extraction.text import CountVectorizer
count_vect=CountVectorizer()
X_train_tf=count_vect.fit_transform(news_train.data)
print(X_train_tf)
X_train_tf.shape
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer=TfidfTransformer()
X_train_tfidf=tfidf_transformer.fit_transform(X_train_tf)
X_train_tfidf.shape
from sklearn.naive_bayes import MultinomialNB
clf= MultinomialNB().fit(X_train_tfidf, news_train.target)
X_test_tf=count_vect.transform(news_test.data)
X_test_tfidf=tfidf_transformer.transform(X_test_tf)
predicted=clf.predict(X_test_tfidf)
predicted
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

```
print("Accuracy",accuracy_score(news_test.target,predicted))
print(metrics.confusion_matrix(news_test.target,predicted))
```

Output:

```
Accuracy 0.8348868175765646
[[192   2   6 119]
 [  2 347   4  36]
 [  2  11 322  61]
 [  2   2   1 393]]
```

LAB PROGRAM 5:

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

Code:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv('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=
BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdiseas
e'),('heartdisease','restecg'),('heartdisease','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={'restecg':1})
print(q1)
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

Output:

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

LAB PROGRAM 6:

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

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)
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]]
```

```
Text(0, 0.5, 'Petal Width')
```

LAB PROGRAM 7:

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:

```
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
import numpy as np
import pandas as pd
iris=datasets.load_iris()
X=iris.data
Y=iris.target
#Shuffle of Data
X,Y = shuffle(X,Y)
model=KMeans(n_clusters=3,init='k-means++',max_iter=10,n_init=1,random_state=3425)
#Training of the model
model.fit(X)
# This is what KMeans thought (Prediction)
Y_Pred=model.labels_
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(Y,Y_Pred)
print(cm)
from sklearn.metrics import accuracy_score
print(accuracy_score(Y,Y_Pred))
#Defining EM Model
from sklearn.mixture import GaussianMixture
model2=GaussianMixture(n_components=3,random_state=3425)
```

#Training of the model

model2.fit(X)

#Predicting classes for our data

Y_predict2= model2.predict(X)

#Accuracy of EM Model

from sklearn.metrics import confusion_matrix

cm=confusion_matrix(Y,Y_predict2)

print(cm)

from sklearn.metrics import accuracy_score

print(accuracy_score(Y,Y_predict2))

Output:

```
#Predicting classes for our data
Y_predict2= model2.predict(X)

#Accuracy of EM Model
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(Y,Y_predict2)
print(cm)

from sklearn.metrics import accuracy_score

print(accuracy_score(Y,Y_predict2))
```

```
[[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
0.36666666666666664
```

LAB PROGRAM 8:

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('target')
print(Y)
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.3)
classier = KNeighborsClassifier(n_neighbors=5)
classier.fit(x_train, y_train)
y_pred=classier.predict(x_test)
print('confusion matrix')
print(confusion_matrix(y_test,y_pred))
print('accuracy')
print(classification_report(y_test,y_pred))
```

Output:

```
print('confusion matrix')
print(confusion_matrix(y_test,y_pred))
```

```
confusion matrix
[[16  0  0]
 [ 0 12  1]
 [ 0  0 16]]
```

```
print('accuracy')
print(classification_report(y_test,y_pred))
```

```
accuracy
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.92      0.96        13
           2       0.94      1.00      0.97        16

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

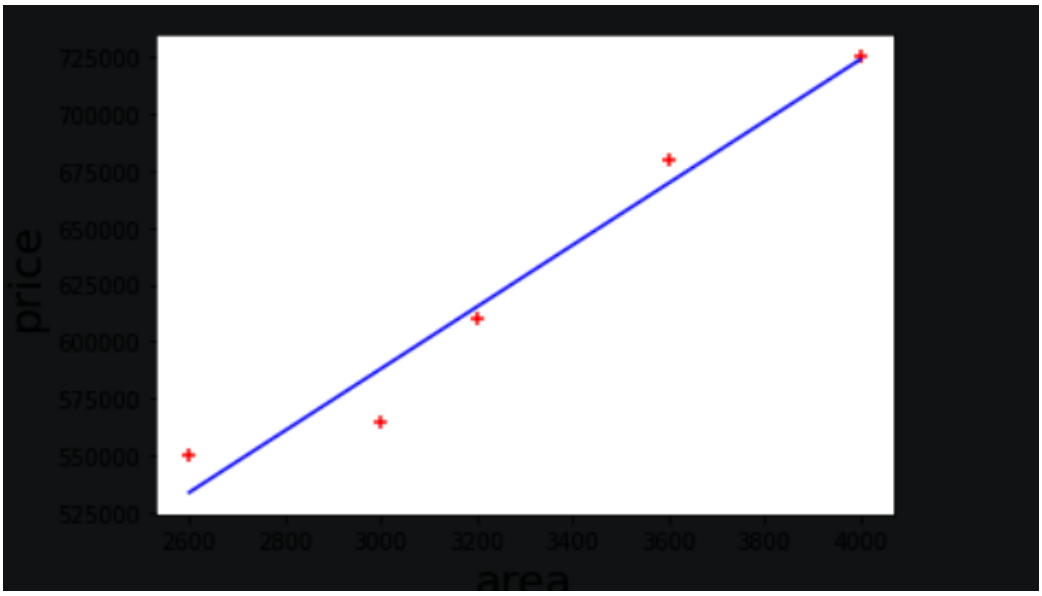
LAB PROGRAM 9:

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

Code:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
df = pd.read_csv(r'/content/homeprices.csv')
df
%matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='.')
new_df = df.drop('price',axis='columns')
df
Price = df.price
Price
reg = linear_model.LinearRegression()
reg.fit(new_df,Price)
reg.predict([[3300]])
#reg.coef_
reg.predict([[5000]])
plt.xlabel('area',fontsize=20)
plt.ylabel('price',fontsize=20)
plt.scatter(df.area,df.price,color='red',marker='+')
plt.plot(df.area,reg.predict(df[['area']]),color='blue')
```

Output:


725000
700000
675000
650000
625000
600000
575000
550000
525000
price
2600
2800
3000
3200
3400
3600
3800
4000
area

LAB PROGRAM 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.

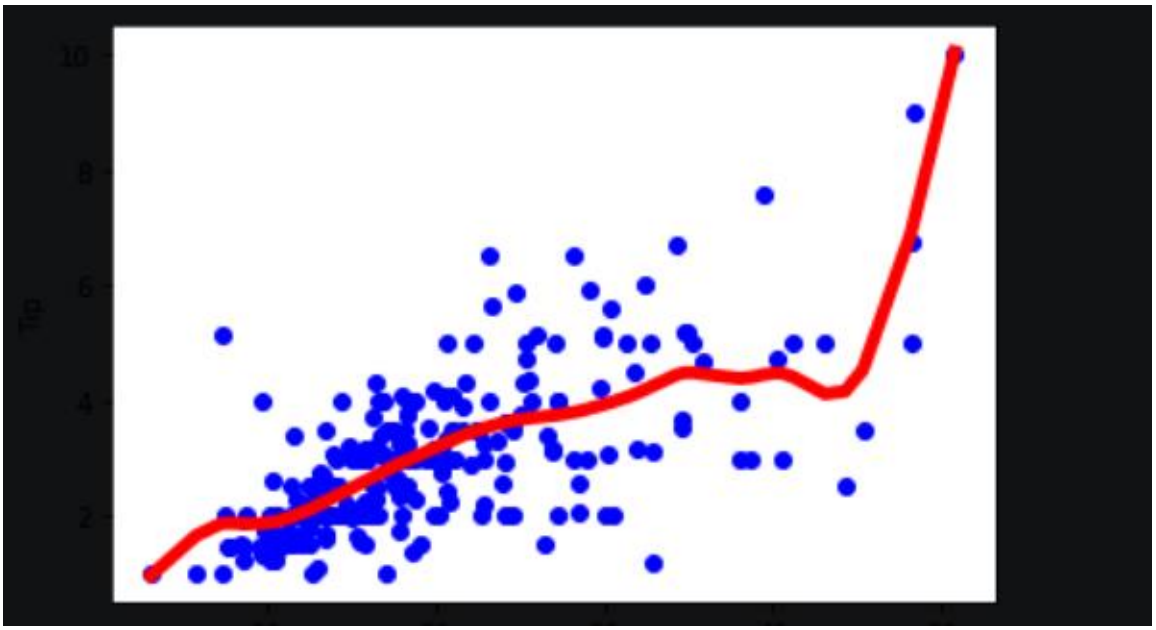
Code:

```
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
data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

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]

one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE

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):
    x0 = np.r_[1, x0]
    X = np.c_[np.ones(len(X)), X]

    xw = X.T * radial_kernel(x0, X, tau)
    beta = np.linalg.pinv(xw @ X) @ xw @ Y
    return x0 @ beta

def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))


n = 1000
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])
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 = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot
```

```
show(gridplot([
[plot_lwr(10.), plot_lwr(1.)],
[plot_lwr(0.1), plot_lwr(0.01)]]))
```

Output:




```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.963963
96
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-2.98256634 -2.99368144 -3.05914505 -3.03174286 -3.07963801 -2.859540
46
 -2.92988067 -2.958209   -2.96962333]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.879598
66
 -2.85953177 -2.83946488 -2.81939799]
```