ML LAB PROGRAMS

PROGRAM 1

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

```
import random
import csv
def read_data(filename):
    with open(filename,"r")as csvfile:
        datareader=csv.reader(csvfile,delimiter=",")
        traindata=[]
        for row in datareader:
            traindata.append(row)
        return (traindata)

#function for finding maximally specific set
def findS():
    dataarr=read_data('enjoysport.csv')
    h=['0','0','0','0','0','0']
    rows=len(dataarr)
    columns=7   #cols=lem(dataarr[0])
    for x in range (1,rows):
        t=dataarr[x]
    #    print(t)
        if t[columns-1]=='1':
            for y in range(0,columns-1):
                if h[y]==t[y]:
                    pass
                elif h[y]!=t[y] and h[y]=='0':
                    h[y]=t[y]
```

```
            elif h[y]!=t[y] and h[y]!='0':
                h[y]='?'
        print(h)
    print("Maximally specific set \n <",end=' ')
    for i in range(0,len(h)):
        print(h[i],end=' ')
        if i!=(len(h)-1) :
            print(",", end='')
    print('>')
findS()
```

enjoysport.csv

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	1
Sunny	Warm	High	Strong	Warm	Same	1
Rainy	Cold	High	Strong	Warm	Change	0
Sunny	Warm	High	Strong	Cool	Change	1

Output

```
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', '?', '?']
Maximally specific set
 < Sunny ,Warm ,? ,Strong ,? ,? >
```

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.

```
import csv
with open('enjoysport.csv') as cfile:
    examples = [ tuple(row) for row in csv.reader(cfile) ]

#lists all unique values of attributes in sorted order
def getdomains(examples):
    d = [ set() for i in examples[0] ]
    for r in examples:
        for c, v in enumerate(r):
            d[c].add(v)
    return [ list(sorted(x)) for x in d ]

def g0(n):
    print('?'*n)
    return ('?',)*n

def s0(n):
    return ('0',)*n

def more_general(h, e):
    mgparts = []
    for x, y in zip(h, e):
        mg = x == '?' or (x != '0' and (x == y or y == "0"))
        mgparts.append(mg)
    return all(mgparts)

def consistent(hypothesis, example):
    return more_general(hypothesis, example)

def min_generalizations(s, e):
    s_new = list(s)
```

```
    for i in range(len(s)):
        if not consistent(s[i:i+1],e[i:i+1]):
            if s[i] != '0':
                s_new[i] = '?'
            else:
                s_new[i] = e[i]
    return [tuple(s_new)]

def generalize_S(e, G, S):
    S_prev = list(S)
    for s in S_prev:
        if not consistent(s,e):
            S.remove(s)
            Splus = min_generalizations(s, e)
            S.update( [ h for h in Splus    if any( [ more_general(g, h)    for g in G ] ) ] )
            S.difference_update( [ h for h in S if any( [ more_general(h, h1) for h1 in S if h != h1 ] ) ]
)
    return S

def min_specializations(h, domains, e):
      results = []
      for i in range(len(h)):
        if h[i] == '?':
          for val in domains[i]:
              if e[i] != val:
                 h_new = h[:i] + (val, ) + h[i+1:]
                 results.append(h_new)
              elif h[i] != '0':
                 h_new = h[: i] + ('0', ) + h[i+1:]
                 results.append(h_new)
      return results
```

```
def specialize_G(e, domains, G, S):
    G_prev = list(G)
    for g in G_prev:
        if g not in G:
            continue
        if consistent(g, e):
            G.remove(g)
            Gminus = min_specializations(g, domains, e)
            G.update( [ h for h in Gminus    if any( [ more_general(h, s)    for s in S] ) ] )
            G.difference_update( [ h for h in G
                    if  any( [ more_general(g1, h)    for g1 in G if h != g1 ] ) ] )
    return G

def candidate_elimination(examples):
        domains = getdomains(examples)[:-1]
        G = set( [ g0(len(domains) ) ] )
        S = set( [ s0(len(domains) ) ] )
        i = 0
        print("\nInitially")
        print("G[{0}]:".format(i), G)
        print("S[{0}]:".format(i), S)
        for r in examples:
            i = i+1
            e, t = r[:-1], r[-1]
            if t == '1':
                G = {g for g in G if consistent(g, e)}
                S = generalize_S(e, G, S)
            else:
                S = {s for s in S if not consistent(s, e)}
                G = specialize_G(e, domains, G, S)
```

```
        print("For Training example {0}".format(i))
        print("G[{0}]:".format(i), G)
        print("S[{0}]:".format(i), S)
    return

candidate_elimination(examples)
```

enjoysport.csv

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	1
Sunny	Warm	High	Strong	Warm	Same	1
Rainy	Cold	High	Strong	Warm	Change	0
Sunny	Warm	High	Strong	Cool	Change	1

Output

Initially

G[0]: {('?', '?', '?', '?', '?', '?')}

S[0]: {('0', '0', '0', '0', '0', '0')}

For Training example 1

G[1]: {('Rainy', '?', '?', '?', '?', '?'), ('?', '?', '?', '?', 'Warm', '?'), ('?', '?', '?', '?', '?', 'Change'), ('?', '?', '?', '?', '?', 'Same'), ('?', 'Cold', '?', '?', '?', '?'), ('Sunny', '?', '?', '?', '?', '?'), ('?', '?', 'Normal', '?', '?', '?'), ('?', '?', '?', 'Strong', '?', '?'), ('?', '?', '?', '?', 'Cool', '?'), ('?', '?', 'High', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}

S[1]: {('0', '0', '0', '0', '0', '0')}

For Training example 2

G[2]: {('?', '?', '?', '?', 'Warm', '?'), ('?', '?', '?', '?', '?', 'Same'), ('Sunny', '?', '?', '?', '?', '?'), ('?', '?', 'Normal', '?', '?', '?'), ('?', '?', '?', 'Strong', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}

S[2]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}

For Training example 3

G[3]: {('?', '?', '?', '?', 'Warm', '?'), ('?', '?', '?', '?', '?', 'Same'), ('Sunny', '?', '?', '?', '?', '?'), ('?', '?', '?', 'Strong', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}

S[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

For Training example 4

G[4]: {('?', 'Warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Same'), ('Sunny', '?', '?', '?', '?', '?')}

S[4]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

For Training example 5

G[5]: {('?', 'Warm', '?', '?', '?', '?'), ('Sunny', '?', '?', '?', '?', '?')}

S[5]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')}

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.

```
import numpy as np
import math
from dataloader import read_data
class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""
    def __str__(self):
        return self.attribute
def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)
    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1
```

```
    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)
    return items, dict
def entropy(S):
    items = np.unique(S)
    if items.size == 1:
        return 0
    counts = np.zeros((items.shape[0], 1))
    sums = 0
    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

    for count in counts:
        sums += -1 * count * math.log(count, 2)
        return sums
def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)
    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))
    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)
```

```
    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)
    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]
    return total_entropy / iv
def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node
    gains = np.zeros((data.shape[1] - 1, 1))

    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)
    split = np.argmax(gains)
    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)
    items, dict = subtables(data, split, delete=True)
    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))
    return node
def empty(size):
    s = ""
    for x in range(size):
        s += " "
    return s
def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return
```

```
    print(empty(level), node.attribute)
    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)
metadata, traindata = read_data("tennis.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

dataloader.py

```
import csv
def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)
    return (metadata, traindata)
```

tennis.csv

outlook	temperature	humidity	wind	answer
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes

rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no

Output

```
outlook
 overcast
  b'yes'
 rain
  wind
   b'strong'
    b'no'
   b'weak'
    b'yes'
 sunny
  humidity
   b'high'
    b'no'
   b'normal'
    b'yes'
```

PROGRAM 4

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
import numpy as np
X=np.array(([2,9], [1,5], [3,6]), dtype=float)
y=np.array(([92], [86], [89]), dtype=float)
X=X/np.amax(X,axis=0)
y=y/100

def sigmoid (x):
    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1-x)

epoch=7000
lr=0.1
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp =outinp1+bout
    output = sigmoid(outinp)
```

```
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH* hiddengrad
    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:
 [[0.89938037]
 [0.87355232]
 [0.89609598]]
```

PROGRAM 5

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

```
import numpy as np
import pandas as pd
mush = pd.read_csv("flu.csv")
mush.replace('?',np.nan,inplace=True)
print(len(mush.columns),"columns, after dropping NA,",len(mush.dropna(axis=1).columns))
#drop wherever you have ? the values are not known
mush.dropna(axis=1,inplace=True)
#the first column in dataset is class which is target variable
target = 'flu'
features = mush.columns[mush.columns != target]
classes = mush[target].unique()
test = mush.sample(frac=0.3)
mush = mush.drop(test.index)
probs = {}
probcl = {}
for x in classes:
    mushcl = mush[mush[target]==x][features]
    clsp = {}
    tot = len(mushcl)
    for col in mushcl.columns:
        colp = {}
        for val,cnt in mushcl[col].value_counts().iteritems():
            #df = pd.DataFrame({'mycolumn': [1,2,2,2,3,3,4]})
             #for val, cnt in df.mycolumn.value_counts().iteritems():
            # print 'value', val, 'was found', cnt, 'times'
             # value 2 was found 3 times
             #value 3 was found 2 times
              #value 4 was found 1 times
               #value 1 was found 1 times
```

```
            pr = cnt/tot
            colp[val] = pr
            clsp[col] = colp

    probs[x] = clsp
    probcl[x] = len(mushcl)/len(mush)

def probabs(x):
    #X - pandas Series with index as feature
    #print("prob")
    if not isinstance(x,pd.Series):
        raise IOError("Arg must of type Series")
    probab = {}
    for cl in classes:
        pr = probcl[cl]
        for col,val in x.iteritems():
            try:
                pr *= probs[cl][col][val]
            except KeyError:
                pr = 0
        probab[cl] = pr
    return probab
def classify(x):
    probab = probabs(x)
    mx = 0
    mxcl = ''
    for cl,pr in probab.items():
        if pr > mx:
            mx = pr
            mxcl = cl
    return mxcl
```

```
#Train data
b = []
for i in mush.index:
  # print(classify(mush.loc[i,features]),mush.loc[i,target])
   b.append(classify(mush.loc[i,features]) == mush.loc[i,target])
print(sum(b),"correct of",len(mush))
print("Accuracy:", sum(b)/len(mush))
#Test data
b = []
for i in test.index:
   #print(classify(mush.loc[i,features]),mush.loc[i,target])
   b.append(classify(test.loc[i,features]) == test.loc[i,target])
print(sum(b),"correct of",len(test))
print("Accuracy:",sum(b)/len(test))
```

flu.csv

flu	chills	runnynose	headache	fever
n	y	n	mild	y
y	y	y	no	n
y	y	n	strong	y
y	n	y	mild	y
n	n	n	no	n
y	n	y	strong	y
n	n	y	strong	n
y	y	y	mild	y

Output

5 columns, after dropping NA, 5

6 correct of 6

Accuracy: 1.0

0 correct of 2

Accuracy: 0.0

PROGRAM 6

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

```
import pandas as pd
msg=pd.read_csv('naivetrext1.csv',names=['message','label'])
#print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
#print(xtest.shape)
#print(xtrain.shape)
#print(ytest.shape)
#print(ytrain.shape)
#print("train data")
#print(xtrain)
#output of count vectoriser is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print(count_vect.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df)#tabular representation
#print(xtrain_dtm) #sparse matrix representation
# 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 metrics
from sklearn import metrics
print('Accuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('Recall and Precison ')
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))
'''docs_new = ['I like this place', 'My boss is not my saviour']
X_new_counts = count_vect.transform(docs_new)
predictednew = clf.predict(X_new_counts)
for doc, category in zip(docs_new, predictednew):
print('%s->%s' % (doc, msg.labelnum[category]))'''
```

naivetrest1.csv

I love this sandwich	pos
This is an amazing place	pos
I feel very good about these beers	pos
This is my best work	pos
What an awesome view	pos
I do not like this restaurant	neg
I am tired of this stuff	neg
I can't deal with this	neg

He is my sworn enemy	neg
My boss is horrible	neg
This is an awesome place	pos
I do not like the taste of this juice	neg
I love to dance	pos
I am sick and tired of this place	neg
What a great holiday	pos
That is a bad locality to stay	neg
We will have good fun tomorrow	pos
I went to my enemy's house today	neg

Output

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

```
   about  am  amazing  an  awesome  beers  best  can  dance  deal  ...  \
0      0   0        0   0        0      0     0    0      1     0  ...
1      0   0        0   0        0      0     0    0      0     0  ...
2      0   0        0   0        0      0     0    0      0     0  ...
3      0   0        0   0        0      0     0    0      0     0  ...
4      1   0        0   0        0      1     0    0      0     0  ...
5      0   0        0   1        1      0     0    0      0     0  ...
6      0   0        0   0        0      0     0    0      0     0  ...
7      0   0        0   0        0      0     1    0      0     0  ...
8      0   0        0   1        1      0     0    0      0     0  ...
9      0   0        1   1        0      0     0    0      0     0  ...
```

```
10     0  1      0  0      0     0    0    0     0    0  ...
11     0  0      0  0      0     0    0    0     0    0  ...
12     0  0      0  0      0     0    0    1     0    1  ...

    this  tired  to  today  very  view  went  what  with  work
0     0     0  1     0     0     0     0     0     0     0
1     1     0  0     0     0     0     0     0     0     0
2     1     0  0     0     0     0     0     0     0     0
3     0     0  1     1     0     0     1     0     0     0
4     0     0  0     0     1     0     0     0     0     0
5     1     0  0     0     0     0     0     0     0     0
6     1     0  0     0     0     0     0     0     0     0
7     1     0  0     0     0     0     0     0     0     1
8     0     0  0     0     0     1     0     1     0     0
9     1     0  0     0     0     0     0     0     0     0
10    1     1  0     0     0     0     0     0     0     0
11    0     0  0     0     0     0     0     1     0     0
12    1     0  0     0     0     0     0     0     1     0

[13 rows x 41 columns]
Accuracy metrics
Accuracy of the classifer is 0.4
Confusion matrix
[[1 3]
 [0 1]]
Recall and Precison
1.0
0.25
```

PROGRAM 7

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

```
from pomegranate import *
asia=DiscreteDistribution({'True' : 0.5, 'False' :0.5})
tuberculosis=ConditionalProbabilityTable(
        [['True','True',0.2],
        ['True','False',0.8],
        ['False','True',0.01],
        ['False','False',0.99]],[asia])
smoking=DiscreteDistribution({'True':0.5,'False':0.5})
lung=ConditionalProbabilityTable(
        [['True','True',0.75],
        ['True','False',0.25],
        ['False','True',0.02],
        ['False','False',0.98]],[smoking])
bronchitis=ConditionalProbabilityTable(
        [['True','True',0.92],
        ['True','False',0.08],
        ['False','True',0.03],
        ['False','False',0.97]],[smoking])
tuberculosis_or_cancer=ConditionalProbabilityTable(
        [['True','True','True',1.0],
        ['True','True','False',0.0],
        ['True','False','True',1.0],
        ['True','False','False',0.0],
        ['False','True','True',1.0],
        ['False','True','False',0.0],
        ['False','False','True',0.0],
```

```
        ['False','False','False',1.0]],[tuberculosis,lung])
xray=ConditionalProbabilityTable(
        [['True','True',0.885],
        ['True','False',0.115],
        ['False','True',0.04],
        ['False','False',0.96]],[tuberculosis_or_cancer])
dyspnea=ConditionalProbabilityTable(
        [['True','True','True',0.96],
        ['True','True','False',0.04],
        ['True','False','True',0.89],
        ['True','False','False',0.11],
        ['False','True','True',0.96],
        ['False','True','False',0.04],
        ['False','False','True',0.89],
        ['False','False','False',0.11]],[tuberculosis_or_cancer,bronchitis])
s0=State(asia,name="asia")
s1=State(tuberculosis,name="tuberculosis")
s2=State(smoking,name="smoker")
network=BayesianNetwork("asia")
network.add_nodes(s0,s1,s2)
network.add_edge(s0,s1)
network.add_edge(s1,s2)
network.bake()
print(network.predict_proba({'tuberculosis':'True'}))
```

Output

```
[{
   "class" :"Distribution",
   "dtype" :"str",
   "name" :"DiscreteDistribution",
   "parameters" :[
```

```
        {
            "True" :0.9523809523809521,
            "False" :0.04761904761904782
        }
    ],
    "frozen" :false
}
 'True'
 {
    "class" :"Distribution",
    "dtype" :"str",
    "name" :"DiscreteDistribution",
    "parameters" :[
        {
            "True" :0.5,
            "False" :0.5
        }
    ],
    "frozen" :false
}]
```

PROGRAM 8

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
from copy import deepcopy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

```
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
# Importing the dataset
data = pd.read_csv('ex.csv')
print("Input Data and Shape")
print(data.shape)
data.head()
print(data.head())
# Getting the values and plotting it
f1 = data['V1'].values
print("f1")
print(f1)
f2 = data['V2'].values
X = np.array(list(zip(f1, f2)))
print("x")
print(X)
print('Graph for whole dataset')
plt.scatter(f1, f2, c='black', s=600)
plt.show()
############################################
kmeans = KMeans(2, random_state=0)
labels = kmeans.fit(X).predict(X)
print("labels")
print(labels)
centroids = kmeans.cluster_centers_
print("centroids")
print(centroids)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40);
print('Graph using Kmeans Algorithm')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, c='#050505')
plt.show()
```

```
#gmm demo
gmm = GaussianMixture(n_components=2).fit(X)
labels = gmm.predict(X)
print("lLABELS GMM")
print(labels)
probs = gmm.predict_proba(X)
size = 10 * probs.max(1) ** 3
print('Graph using EM Algorithm')
#print(probs[:300].round(4))
plt.scatter(X[:, 0], X[:, 1], c=labels, s=size, cmap='viridis');
plt.show()
```

ex.csv

"	V1	V2
1	1	1
2	1.5	2
3	3	4
4	5	7
5	3.5	5
6	4.5	5
7	3.5	4.5

Output

```
Input Data and Shape
(7, 3)
   "  V1   V2
0  1  1.0  1.0
1  2  1.5  2.0
2  3  3.0  4.0
3  4  5.0  7.0
4  5  3.5  5.0
```

f1

[1. 1.5 3. 5. 3.5 4.5 3.5]

x

[[1. 1.]

 [1.5 2.]

 [3. 4.]

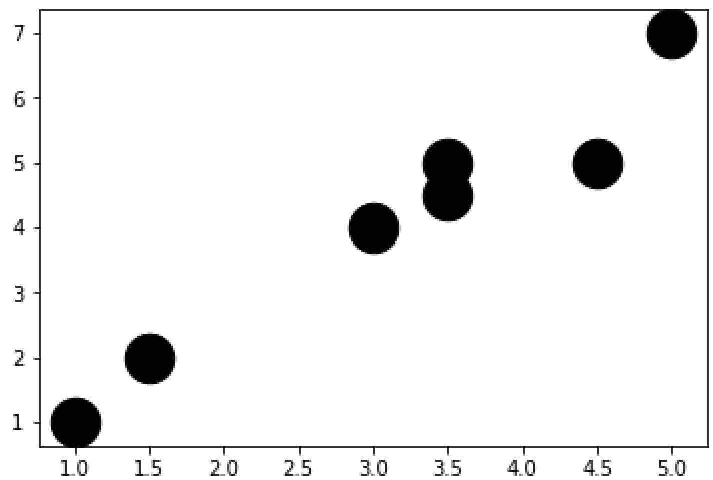
 [5. 7.]

 [3.5 5.]

 [4.5 5.]

 [3.5 4.5]]

Graph for whole dataset



labels

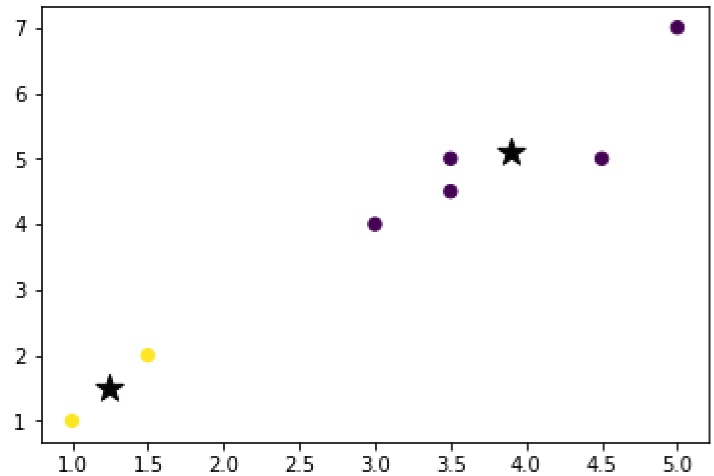
[1 1 0 0 0 0 0]

centroids

[[3.9 5.1]

 [1.25 1.5]]

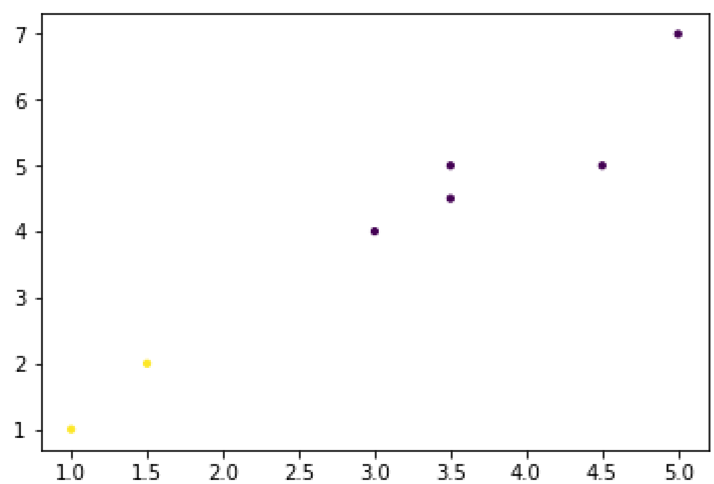
Graph using Kmeans Algorithm



lLABELS GMM

[1 1 0 0 0 0 0]

Graph using EM Algorithm



PROGRAM 9

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem..

from sklearn.datasets import load_iris

```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split
iris_dataset=load_iris()
print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris_dataset.target_names)
for i in range(len(iris_dataset.target_names)):
    print("\n[{0}]:[{1}]".format(i,iris_dataset.target_names[i]))

print("\n IRIS DATA :\n",iris_dataset["data"])

X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"],random_state=0)

print("\n Target :\n",iris_dataset["target"])
print("\n X TRAIN \n", X_train)
print("\n X TEST \n", X_test)
print("\n Y TRAIN \n", y_train)
print("\n Y TEST \n", y_test)
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)

for i in range(len(X_test)):
    x = X_test[i]
    x_new = np.array([x])
    prediction = kn.predict(x_new)

    print("\n Actual : {0} {1}, Predicted:{2}{3}".format(y_test[i],iris_dataset["target_names"][y_test[i]],prediction,iris_dataset["target_names"][prediction]))

print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X_test, y_test)))
```

Output

IRIS FEATURES \ TARGET NAMES:

['setosa' 'versicolor' 'virginica']

[0]:[setosa]

[1]:[versicolor]

[2]:[virginica]

IRIS DATA :

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
```

```
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.1 1.5 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
```

```
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
```

```
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
```

```
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
```

[6.7 3. 5.2 2.3]
[6.3 2.5 5. 1.9]
[6.5 3. 5.2 2.]
[6.2 3.4 5.4 2.3]
[5.9 3. 5.1 1.8]]

Target :
[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]

X TRAIN
[[5.9 3. 4.2 1.5]
[5.8 2.6 4. 1.2]
[6.8 3. 5.5 2.1]
[4.7 3.2 1.3 0.2]
[6.9 3.1 5.1 2.3]
[5. 3.5 1.6 0.6]
[5.4 3.7 1.5 0.2]
[5. 2. 3.5 1.]
[6.5 3. 5.5 1.8]
[6.7 3.3 5.7 2.5]
[6. 2.2 5. 1.5]
[6.7 2.5 5.8 1.8]
[5.6 2.5 3.9 1.1]
[7.7 3. 6.1 2.3]
[6.3 3.3 4.7 1.6]
[5.5 2.4 3.8 1.1]
[6.3 2.7 4.9 1.8]

```
[6.3 2.8 5.1 1.5]
[4.9 2.5 4.5 1.7]
[6.3 2.5 5.  1.9]
[7.  3.2 4.7 1.4]
[6.5 3.  5.2 2. ]
[6.  3.4 4.5 1.6]
[4.8 3.1 1.6 0.2]
[5.8 2.7 5.1 1.9]
[5.6 2.7 4.2 1.3]
[5.6 2.9 3.6 1.3]
[5.5 2.5 4.  1.3]
[6.1 3.  4.6 1.4]
[7.2 3.2 6.  1.8]
[5.3 3.7 1.5 0.2]
[4.3 3.  1.1 0.1]
[6.4 2.7 5.3 1.9]
[5.7 3.  4.2 1.2]
[5.4 3.4 1.7 0.2]
[5.7 4.4 1.5 0.4]
[6.9 3.1 4.9 1.5]
[4.6 3.1 1.5 0.2]
[5.9 3.  5.1 1.8]
[5.1 2.5 3.  1.1]
[4.6 3.4 1.4 0.3]
[6.2 2.2 4.5 1.5]
[7.2 3.6 6.1 2.5]
[5.7 2.9 4.2 1.3]
[4.8 3.  1.4 0.1]
[7.1 3.  5.9 2.1]
[6.9 3.2 5.7 2.3]
[6.5 3.  5.8 2.2]
```

```
[6.4 2.8 5.6 2.1]
[5.1 3.8 1.6 0.2]
[4.8 3.4 1.6 0.2]
[6.5 3.2 5.1 2. ]
[6.7 3.3 5.7 2.1]
[4.5 2.3 1.3 0.3]
[6.2 3.4 5.4 2.3]
[4.9 3.  1.4 0.2]
[5.7 2.5 5.  2. ]
[6.9 3.1 5.4 2.1]
[4.4 3.2 1.3 0.2]
[5.  3.6 1.4 0.2]
[7.2 3.  5.8 1.6]
[5.1 3.5 1.4 0.3]
[4.4 3.  1.3 0.2]
[5.4 3.9 1.7 0.4]
[5.5 2.3 4.  1.3]
[6.8 3.2 5.9 2.3]
[7.6 3.  6.6 2.1]
[5.1 3.5 1.4 0.2]
[4.9 3.1 1.5 0.1]
[5.2 3.4 1.4 0.2]
[5.7 2.8 4.5 1.3]
[6.6 3.  4.4 1.4]
[5.  3.2 1.2 0.2]
[5.1 3.3 1.7 0.5]
[6.4 2.9 4.3 1.3]
[5.4 3.4 1.5 0.4]
[7.7 2.6 6.9 2.3]
[4.9 2.4 3.3 1. ]
[7.9 3.8 6.4 2. ]
```

```
[6.7 3.1 4.4 1.4]
[5.2 4.1 1.5 0.1]
[6.  3.  4.8 1.8]
[5.8 4.  1.2 0.2]
[7.7 2.8 6.7 2. ]
[5.1 3.8 1.5 0.3]
[4.7 3.2 1.6 0.2]
[7.4 2.8 6.1 1.9]
[5.  3.3 1.4 0.2]
[6.3 3.4 5.6 2.4]
[5.7 2.8 4.1 1.3]
[5.8 2.7 3.9 1.2]
[5.7 2.6 3.5 1. ]
[6.4 3.2 5.3 2.3]
[6.7 3.  5.2 2.3]
[6.3 2.5 4.9 1.5]
[6.7 3.  5.  1.7]
[5.  3.  1.6 0.2]
[5.5 2.4 3.7 1. ]
[6.7 3.1 5.6 2.4]
[5.8 2.7 5.1 1.9]
[5.1 3.4 1.5 0.2]
[6.6 2.9 4.6 1.3]
[5.6 3.  4.1 1.3]
[5.9 3.2 4.8 1.8]
[6.3 2.3 4.4 1.3]
[5.5 3.5 1.3 0.2]
[5.1 3.7 1.5 0.4]
[4.9 3.1 1.5 0.1]
[6.3 2.9 5.6 1.8]
[5.8 2.7 4.1 1. ]
```

[7.7 3.8 6.7 2.2]
[4.6 3.2 1.4 0.2]]

X TEST
[[5.8 2.8 5.1 2.4]
[6. 2.2 4. 1.]
[5.5 4.2 1.4 0.2]
[7.3 2.9 6.3 1.8]
[5. 3.4 1.5 0.2]
[6.3 3.3 6. 2.5]
[5. 3.5 1.3 0.3]
[6.7 3.1 4.7 1.5]
[6.8 2.8 4.8 1.4]
[6.1 2.8 4. 1.3]
[6.1 2.6 5.6 1.4]
[6.4 3.2 4.5 1.5]
[6.1 2.8 4.7 1.2]
[6.5 2.8 4.6 1.5]
[6.1 2.9 4.7 1.4]
[4.9 3.1 1.5 0.1]
[6. 2.9 4.5 1.5]
[5.5 2.6 4.4 1.2]
[4.8 3. 1.4 0.3]
[5.4 3.9 1.3 0.4]
[5.6 2.8 4.9 2.]
[5.6 3. 4.5 1.5]
[4.8 3.4 1.9 0.2]
[4.4 2.9 1.4 0.2]
[6.2 2.8 4.8 1.8]
[4.6 3.6 1. 0.2]
[5.1 3.8 1.9 0.4]

```
[6.2 2.9 4.3 1.3]
[5.  2.3 3.3 1. ]
[5.  3.4 1.6 0.4]
[6.4 3.1 5.5 1.8]
[5.4 3.  4.5 1.5]
[5.2 3.5 1.5 0.2]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.2]
[5.2 2.7 3.9 1.4]
[5.7 3.8 1.7 0.3]
[6.  2.7 5.1 1.6]]

Y TRAIN
[1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0 1
0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0 0
1 0 2 1 2 1 0 2 0 2 0 0 2 0 2 1 1 1 2 2 1 1 0 1 2 2 0 1 1 1 1 0 0 0 2 1 2
0]

Y TEST
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
1]

Actual : 2 virginica, Predicted:[2]['virginica']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 2 virginica, Predicted:[2]['virginica']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 2 virginica, Predicted:[2]['virginica']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 1 versicolor, Predicted:[1]['versicolor']
```

Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 2 virginica, Predicted:[2]['virginica']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 2 virginica, Predicted:[2]['virginica']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 2 virginica, Predicted:[2]['virginica']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 2 virginica, Predicted:[2]['virginica']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 2 virginica, Predicted:[2]['virginica']
Actual : 2 virginica, Predicted:[2]['virginica']
Actual : 1 versicolor, Predicted:[1]['versicolor']
Actual : 0 setosa, Predicted:[0]['setosa']
Actual : 1 versicolor, Predicted:[2]['virginica']
TEST SCORE[ACCURACY]: 0.97

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.

```
from math import ceil
import numpy as np
from scipy import linalg
def lowess(x, y, f= 2. / 3., iter=3):
    n = len(x) # Number of x  points
    r = int(ceil(f * n))  # Computing the residual of smoothing functions
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)] #
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)  # Weight Function
    w = (1 - w ** 3) ** 3  # Tricube Weight Function
    ypred = np.zeros(n) # Initialisation of predictor
    delta = np.ones(n)  # Initialisation of delta
    for iteration in range(iter):
        for i in range(n):
            weights = delta * w[:, i] # Cumulative Weights
            b = np.array([np.sum(weights * y), np.sum(weights * y * x)]) # Matrix B
            A = np.array([[np.sum(weights), np.sum(weights * x)],
                          [np.sum(weights * x), np.sum(weights * x * x)]]) # Matrix A
            beta = linalg.solve(A, b) # Beta,Solution of AX= B equation
            ypred[i] = beta[0] + beta[1] * x[i]
        residuals = y - ypred   # Finding Residuals
        s = np.median(np.abs(residuals))  # Median of Residuals
        delta = np.clip(residuals / (6.0 * s), -1, 1)  # Delta
        delta = (1 - delta ** 2) ** 2   # Delta
    return ypred

if __name__ == '__main__':  # Main Function
    import math
```

```
n = 100  # Number of data points
#Case1: Sinusoidal Fitting
x = np.linspace(0, 2 * math.pi, n)
print(x)
y = np.sin(x) + 0.3 * np.random.randn(n)
  #Case2 : Straight Line Fitting
#x=np.linspace(0,2.5,n) # For Linear
#y= 1 + 0.25*np.random.randn(n) # For Linear
f = 0.25
ypred = lowess(x, y, f=f, iter=3)
import pylab as pl
pl.clf()
pl.plot(x, y, label='Y NOISY')
pl.plot(x, ypred, label='Y PREDICTED')
pl.legend()
pl.show()
```

Output

```
[0.         0.06346652 0.12693304 0.19039955 0.25386607 0.31733259
 0.38079911 0.44426563 0.50773215 0.57119866 0.63466518 0.6981317
 0.76159822 0.82506474 0.88853126 0.95199777 1.01546429 1.07893081
 1.14239733 1.20586385 1.26933037 1.33279688 1.3962634  1.45972992
 1.52319644 1.58666296 1.65012947 1.71359599 1.77706251 1.84052903
 1.90399555 1.96746207 2.03092858 2.0943951  2.15786162 2.22132814
 2.28479466 2.34826118 2.41172769 2.47519421 2.53866073 2.60212725
 2.66559377 2.72906028 2.7925268  2.85599332 2.91945984 2.98292636
 3.04639288 3.10985939 3.17332591 3.23679243 3.30025895 3.36372547
 3.42719199 3.4906585  3.55412502 3.61759154 3.68105806 3.74452458
 3.8079911  3.87145761 3.93492413 3.99839065 4.06185717 4.12532369
 4.1887902  4.25225672 4.31572324 4.37918976 4.44265628 4.5061228
```

4.56958931 4.63305583 4.69652235 4.75998887 4.82345539 4.88692191
4.95038842 5.01385494 5.07732146 5.14078798 5.2042545 5.26772102
5.33118753 5.39465405 5.45812057 5.52158709 5.58505361 5.64852012
5.71198664 5.77545316 5.83891968 5.9023862 5.96585272 6.02931923
6.09278575 6.15625227 6.21971879 6.28318531]